



Uleam

Aplicaciones Web I

**Tema: Software de gestión para el
Departamento de Talento Humano de la
ULEAM**

Nombre:

Fuentes Cedeño José Miguel

1. Descripción del negocio o tema elegido.

El tema elegido es la gestión interna de capacitaciones y la publicación de anuncios dirigidos específicamente al personal administrativo del Departamento de Talento Humano de la Universidad Laica Eloy Alfaro de Manabí (ULEAM), ubicada en Manta, Manabí, Ecuador. El dominio de esta aplicación se centra en mejorar la comunicación interna, la organización de eventos de capacitación y la optimización del acceso a la información relevante para el personal.

Actualmente, la ULEAM busca modernizar y centralizar los procesos relacionados con el desarrollo profesional de su personal. Este sistema servirá como una plataforma para que los administradores del Departamento de Talento Humano puedan:

- Crear y gestionar un catálogo de capacitaciones disponibles.
- Registrar la inscripción de usuarios a dichas capacitaciones.
- Emitir y gestionar certificados de asistencia o culminación.
- Publicar y mantener un repositorio de anuncios y novedades importantes.

Por otro lado, los empleados de la ULEAM tendrán un espacio para visualizar su perfil, ver las capacitaciones disponibles, revisar su historial de inscripciones, y acceder a los certificados obtenidos, así como mantenerse informados a través de los anuncios publicados.

2. Propósito del sistema web.

El propósito principal de este sistema web es optimizar y digitalizar la administración de capacitaciones y la difusión de información relevante dentro del Departamento de Talento Humano de la ULEAM. Busca centralizar estos procesos para mejorar la eficiencia operativa, asegurar que el personal esté debidamente capacitado y bien informado, y proporcionar un acceso fácil y rápido a sus registros personales de capacitación y certificaciones. En última instancia, el sistema tiene como objetivo contribuir al desarrollo profesional continuo del personal administrativo y fortalecer la comunicación interna.

3. Requerimientos funcionales y no funcionales.

Requerimientos Funcionales:

- **Acceso por Rol:** Autenticación para usuarios (empleados) y administradores (Talento Humano).
- **Gestión de Capacitaciones (Admin):** Crear, editar y eliminar capacitaciones; gestionar inscripciones de usuarios.
- **Gestión de Anuncios (Admin):** Crear, editar y eliminar anuncios con contenido enriquecido.
- **Perfil de Usuario (Empleado):** Visualizar información personal, capacitaciones inscritas y certificados obtenidos.
- **Visualización de Anuncios (Empleado):** Acceder a todos los comunicados publicados.
- **Cierre de Sesión:** Funcionalidad segura para terminar la sesión.

Requerimientos No Funcionales:

- **Rendimiento:** Rapidez en la carga de la interfaz y operaciones principales.
- **Seguridad:** Control de acceso basado en roles y protección de datos básicos.
- **Usabilidad:** Interfaz intuitiva y fácil de usar.
- **Responsividad:** Adaptación del diseño a diferentes tamaños de pantalla (escritorio, tablet, móvil).
- **Persistencia de Datos:** Los datos deben guardarse para su uso continuo.

4. Herramientas y Tecnologías de Programación y Almacenamiento de Datos.

- **Frontend:** React.js (con JavaScript), React Router DOM para navegación, TinyMCE como editor de texto enriquecido, y CSS3 para estilos.
- **Almacenamiento de Datos:** localStorage del navegador para persistencia de datos del cliente (usuarios, capacitaciones, anuncios, inscripciones).

5. Método de publicación y hosting.

El proyecto es una aplicación web estática (React frontend). Se publica generando archivos optimizados (`npm run dev`) y puede ser alojado fácilmente en servicios de hosting de estáticos como Netlify, Vercel o GitHub Pages.

6. Código fuente completo. (Además, incluir enlace GitLab).

GitLab: <https://gitlab.com/jm5966829/react-localstorage/->

[/blob/c7d26c010e881a35d23d5fb61dfa1d95a2b47615/Proyecto-React-Ls.zip](#)

- **ADMIN**

1. AdminDashboard

```
import React, { useEffect, useState } from 'react';
import { Outlet, useNavigate, Link, useLocation } from 'react-router-dom';
import '../CSS/AdminInterface.css';

const ADMIN_AREA = 'Departamento de Talento Humano';

const AdminDashboard = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const [userName, setUserName] = useState('');
  const [userArea, setUserArea] = useState('');

  useEffect(() => {
    const storedArea = localStorage.getItem('currentUserArea');
    const storedName = localStorage.getItem('currentUserName');

    if (!storedArea || storedArea !== ADMIN_AREA) {
      alert('Acceso denegado. Debes iniciar sesión como administrador.');
```

// Usamos setTimeout para asegurar que la alerta se muestre antes de la redirección

```
      setTimeout(() => {
        navigate('/login');
      }, 0);
    } else {
      setUserName(storedName || 'Administrador');
      setUserArea(storedArea);
    }
  }, [navigate]);

  const handleLogout = (e) => {
    e.preventDefault();
    localStorage.removeItem('currentUserCedula');
    localStorage.removeItem('currentUserName');
    localStorage.removeItem('currentUserArea');
    navigate('/login');
  };

  if (userArea !== ADMIN_AREA) {
    return null;
  }
}
```

```
const isDashboardRoot = location.pathname === '/dashboard';

const mainContentClass = `main-content ${isDashboardRoot ? 'main-content-home-bg' : ''}`;

return (
  <div className="admin-dashboard-wrapper">
    <div className="sidebar">
      <div className="logo">
        <i className="fas fa-cubes"></i> ULEAM
      </div>
      <ul>
        <li>
          <Link
            to="/dashboard"
            className={isDashboardRoot ? 'active' : ''}
          >
            <i className="fas fa-home"></i> Inicio
          </Link>
        </li>
        <li>
          <Link
            to="/dashboard/capacitaciones"
            className={location.pathname === '/dashboard/capacitaciones' ? 'active' : ''}
          >
            <i className="fas fa-chalkboard-teacher"></i> Capacitaciones
          </Link>
        </li>
        <li>
          <Link
            to="/dashboard/anuncios"
            className={location.pathname === '/dashboard/anuncios' ? 'active' : ''}
          >
            <i className="fas fa-bullhorn"></i> Anuncios
          </Link>
        </li>
        <li>
          <Link
            to="/dashboard/certificados"
            className={location.pathname === '/dashboard/certificados' ? 'active' : ''}
          >
            <i className="fas fa-certificate"></i> Certificados
          </Link>
        </li>
        <li>
          <Link
            to="/dashboard/usuarios"
            className={location.pathname === '/dashboard/usuarios' ? 'active' : ''}
          >
            <i className="fas fa-user-circle"></i> Usuarios
          </Link>
        </li>
      </ul>
    </div>
  </div>
);
```

```

</Link>
</li>
<li>
<a href="#" onClick={handleLogout}>
<i className="fas fa-sign-out-alt"></i> Cerrar Sesión
</a>
</li>
</ul>
</div>

<div className={mainContentClass}>
{isDashboardRoot && (
<>
<div className="anuncios-bar">
<div className="anuncio-item">
<span className="icon"><i className="fas fa-bell"></i></span>
<p>Bienvenido, <strong>{ADMIN_AREA}</strong></p>
</div>
</div>
</>
)}

<Outlet />
</div>
</div>
);
};

export default AdminDashboard;

```

2. Anuncios

```

import React, { useEffect, useState } from 'react';
import { Outlet, useNavigate, Link, useLocation } from 'react-router-dom';
import '../CSS/AdminInterface.css';

const ADMIN_AREA = 'Departamento de Talento Humano';

const AdminDashboard = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const [userName, setUserName] = useState('');
  const [userArea, setUserArea] = useState('');

  useEffect(() => {
    const storedArea = localStorage.getItem('currentUserArea');
    const storedName = localStorage.getItem('currentUserName');

    if (!storedArea || storedArea !== ADMIN_AREA) {

```

```

        alert('Acceso denegado. Debes iniciar sesión como administrador.');
```

// Usamos setTimeout para asegurar que la alerta se muestre antes de la redirección

```

        setTimeout(() => {
            navigate('/login');
        }, 0);
    } else {
        setUsername(storedName || 'Administrador');
        setUserArea(storedArea);
    }
}, [navigate]);

const handleLogout = (e) => {
    e.preventDefault();
    localStorage.removeItem('currentUserCedula');
    localStorage.removeItem('currentUserName');
    localStorage.removeItem('currentUserArea');
    navigate('/login');
};

if (userArea !== ADMIN_AREA) {
    return null;
}

const isDashboardRoot = location.pathname === '/dashboard';

const mainContentClass = `main-content ${isDashboardRoot ? 'main-content-home-bg' : ''}`;

return (
    <div className="admin-dashboard-wrapper">
        <div className="sidebar">
            <div className="logo">
                <i className="fas fa-cubes"></i> ULEAM
            </div>
            <ul>
                <li>
                    <Link
                        to="/dashboard"
                        className={isDashboardRoot ? 'active' : ''}
                    >
                        <i className="fas fa-home"></i> Inicio
                    </Link>
                </li>
                <li>
                    <Link
                        to="/dashboard/capacitaciones"
                        className={location.pathname === '/dashboard/capacitaciones' ?
'active' : ''}
                    >
                        <i className="fas fa-chalkboard-teacher"></i> Capacitaciones

```

```

        </Link>
      </li>
      <li>
        <Link
          to="/dashboard/anuncios"
          className={location.pathname === '/dashboard/anuncios' ? 'active'
: ''}
        >
          <i className="fas fa-bullhorn"></i> Anuncios
        </Link>
      </li>
      <li>
        <Link
          to="/dashboard/certificados"
          className={location.pathname === '/dashboard/certificados' ?
'active' : ''}
        >
          <i className="fas fa-certificate"></i> Certificados
        </Link>
      </li>
      <li>
        <Link
          to="/dashboard/usuarios"
          className={location.pathname === '/dashboard/usuarios' ? 'active'
: ''}
        >
          <i className="fas fa-user-circle"></i> Usuarios
        </Link>
      </li>
      <li>
        <a href="#" onClick={handleLogout}>
          <i className="fas fa-sign-out-alt"></i> Cerrar Sesión
        </a>
      </li>
    </ul>
  </div>

  <div className={mainContentClass}>
    {isDashboardRoot && (
      <>
        <div className="anuncios-bar">
          <div className="anuncio-item">
            <span className="icon"><i className="fas fa-bell"></i></span>
            <p>Bienvenido, <strong>{ADMIN_AREA}</strong></p>
          </div>
        </div>
      </div>
    </>
  )}

  <Outlet />

```



```

        </div>
      </div>
    );
  };

export default AdminDashboard;

```

3. Capacitaciones

```

import React, { useState, useRef, useEffect } from 'react';
import { Editor } from '@tinymce/tinymce-react';
import '../CSS/AdminInterface.css';

const Capacitaciones = () => {
  // Referencia para el editor de TinyMCE
  const editorRef = useRef(null);

  // Estado para la lista de capacitaciones
  // Inicializa el estado leyendo de localStorage
  const [capacitaciones, setCapacitaciones] = useState(() => {
    try {
      const storedCapacitaciones = localStorage.getItem('capacitacionesData');
      // Si hay datos en localStorage, parsea y úsalos; de lo contrario, un arreglo
      vacío.
      return storedCapacitaciones ? JSON.parse(storedCapacitaciones) : [];
    } catch (error) {
      console.error("Error al cargar capacitaciones de localStorage:", error);
      return [];
    }
  });

  // useEffect para guardar las capacitaciones en localStorage cada vez que cambian
  useEffect(() => {
    try {
      localStorage.setItem('capacitacionesData', JSON.stringify(capacitaciones));
    } catch (error) {
      console.error("Error al guardar capacitaciones en localStorage:", error);
    }
  }, [capacitaciones]);

  // Estado para controlar la visibilidad del formulario de creación/edición
  const [showForm, setShowForm] = useState(false);

  // Estado para los datos de la nueva capacitación o la que se está editando
  const [currentCapacitacion, setCurrentCapacitacion] = useState({
    id: null,
    titulo: '',
    descripcionCorta: '',
    duracion: '',
    tipoInscripcion: 'Libre', // Valor por defecto
  });

```

```

        fechaInicio: '',
        fechaFin: '',
        contenidoCompleto: '<p>Aquí puedes añadir el temario, enlaces a videos, tareas,
etc.</p>'
    });

    // Función para manejar cambios en los campos del formulario
    const handleChange = (e) => {
        const { name, value } = e.target;
        setCurrentCapacitacion(prev => ({ ...prev, [name]: value }));
    };

    // Función para manejar cambios en el editor TinyMCE
    const handleEditorChange = (content, editor) => {
        setCurrentCapacitacion(prev => ({ ...prev, contenidoCompleto: content }));
    };

    // Función para añadir/actualizar una capacitación
    const handleSubmit = (e) => {
        e.preventDefault();

        // Validaciones básicas
        if (!currentCapacitacion.titulo || !currentCapacitacion.descripcionCorta ||
!currentCapacitacion.fechaInicio || !currentCapacitacion.fechaFin) {
            alert('Por favor, completa todos los campos obligatorios: Título, Descripción
Corta, Fecha Inicio y Fecha Fin.');
```

Corta, Fecha Inicio y Fecha Fin.');

```

            return;
        }

        if (new Date(currentCapacitacion.fechaInicio) > new
Date(currentCapacitacion.fechaFin)) {
            alert('La fecha de fin no puede ser anterior a la fecha de inicio.');
```

La fecha de fin no puede ser anterior a la fecha de inicio.');

```

            return;
        }

        if (currentCapacitacion.id) {
            // Lógica para actualizar una capacitación existente
            setCapacitaciones(prevCaps =>
                prevCaps.map(cap =>
                    cap.id === currentCapacitacion.id ? currentCapacitacion : cap
                )
            );
            alert('Capacitación actualizada con éxito.');
```

Capacitación actualizada con éxito.');

```

        } else {
            // Lógica para añadir una nueva capacitación
            const newCapacitacion = {
                ...currentCapacitacion,
                id: Date.now() // Generar un ID único
            };
            setCapacitaciones(prevCaps => [...prevCaps, newCapacitacion]);
        }
    };

```

```

        alert('Capacitación creada con éxito.');
```

```
    }
```

```
    // Limpiar formulario y ocultar
```

```
    setCurrentCapacitacion({
```

```
        id: null,
```

```
        titulo: '',
```

```
        descripcionCorta: '',
```

```
        duracion: '',
```

```
        tipoInscripcion: 'Libre',
```

```
        fechaInicio: '',
```

```
        fechaFin: '',
```

```
        contenidoCompleto: '<p>Aquí puedes añadir el temario, enlaces a videos, tareas,  
etc.</p>' // Contenido inicial
```

```
    });
```

```
    setShowForm(false);
```

```
};
```

```
// Función para editar una capacitación
```

```
const handleEdit = (cap) => {
```

```
    setCurrentCapacitacion(cap);
```

```
    setShowForm(true);
```

```
};
```

```
// Función para eliminar una capacitación
```

```
const handleDelete = (id) => {
```

```
    if (window.confirm('¿Estás seguro de que deseas eliminar esta capacitación?')) {
```

```
        setCapacitaciones(prevCaps => prevCaps.filter(cap => cap.id !== id));
```

```
        alert('Capacitación eliminada.');
```

```
    }
```

```
};
```

```
// Función para abrir el formulario para crear una nueva
```

```
const handleNewCapacitacion = () => {
```

```
    setCurrentCapacitacion({
```

```
        id: null,
```

```
        titulo: '',
```

```
        descripcionCorta: '',
```

```
        duracion: '',
```

```
        tipoInscripcion: 'Libre',
```

```
        fechaInicio: '',
```

```
        fechaFin: '',
```

```
        contenidoCompleto: '<p>Aquí puedes añadir el temario, enlaces a videos, tareas,  
etc.</p>'
```

```
    });
```

```
    setShowForm(true);
```

```
};
```

```
return (
```

```
    <div className="capacitaciones-section-wrapper">
```

```

<div className="section-header-with-button">
  <h1>Gestión de Capacitaciones</h1>
  <button
    className="btn-add-new"
    onClick={handleNewCapacitacion}
  >
    + Crear Nueva Capacitación
  </button>
</div>

{capacitaciones.length === 0 && !showForm && (
  <div className="empty-state-message">
    <p>Aún no hay capacitaciones disponibles.</p>
  </div>
)}

{/* Tabla de capacitaciones */}
{capacitaciones.length > 0 && (
  <table className="data-table">
    <thead>
      <tr>
        <th>Título</th>
        <th>Descripción Corta</th>
        <th>Duración</th>
        <th>Tipo</th>
        <th>Fecha Inicio</th>
        <th>Fecha Fin</th>
        <th>Acciones</th>
      </tr>
    </thead>
    <tbody>
      {capacitaciones.map(cap => (
        <tr key={cap.id}>
          <td>{cap.titulo}</td>
          <td>{cap.descripcionCorta}</td>
          <td>{cap.duracion}</td>
          <td>{cap.tipoInscripcion}</td>
          <td>{cap.fechaInicio}</td>
          <td>{cap.fechaFin}</td>
          <td className="action-buttons">
            <button className="edit-btn" onClick={() =>
handleEdit(cap)}>Editar</button>
            <button className="delete-btn" onClick={() =>
handleDelete(cap.id)}>Eliminar</button>
          </td>
        </tr>
      )
    )
  </tbody>
</table>
)}

```

```

    { /* Modal/Formulario de Creación/Edición */}
    {showForm && (
      <div className="form-modal-overlay">
        <div className="form-modal-content">
          <h2>{currentCapacitacion.id ? 'Editar Capacitación' : 'Crear Nueva
Capacitación'}</h2>
          <form onSubmit={handleSubmit}>
            <div className="form-group">
              <label htmlFor="titulo">Título:</label>
              <input
                type="text"
                id="titulo"
                name="titulo"
                value={currentCapacitacion.titulo}
                onChange={handleChange}
                required
              />
            </div>
            <div className="form-group">
              <label htmlFor="descripcionCorta">Descripción Corta:</label>
              <textarea
                id="descripcionCorta"
                name="descripcionCorta"
                value={currentCapacitacion.descripcionCorta}
                onChange={handleChange}
                rows="3"
                required
              ></textarea>
            </div>
            <div className="form-group">
              <label htmlFor="duracion">Duración Estimada:</label>
              <input
                type="text"
                id="duracion"
                name="duracion"
                value={currentCapacitacion.duracion}
                onChange={handleChange}
              />
            </div>
            <div className="form-group">
              <label htmlFor="tipoInscripcion">Tipo de Inscripción:</label>
              <select
                id="tipoInscripcion"
                name="tipoInscripcion"
                value={currentCapacitacion.tipoInscripcion}
                onChange={handleChange}
              >
                <option value="Libre">Libre</option>
                <option value="Obligatoria">Obligatoria</option>
              </select>
            </div>
          </form>
        </div>
      </div>
    )}
  )}

```

```

        </select>
    </div>
    <div className="form-group">
        <label htmlFor="fechaInicio">Fecha de Inicio:</label>
        <input
            type="date"
            id="fechaInicio"
            name="fechaInicio"
            value={currentCapitacion.fechaInicio}
            onChange={handleChange}
            required
        />
    </div>
    <div className="form-group">
        <label htmlFor="fechaFin">Fecha de Fin:</label>
        <input
            type="date"
            id="fechaFin"
            name="fechaFin"
            value={currentCapitacion.fechaFin}
            onChange={handleChange}
            required
        />
    </div>

    <div className="form-group tinymce-editor-container">
        <label>Contenido Completo de la Capacitación:</label>
        <Editor
            apiKey='lepi3l25n6c5dwjx7r02sceh34lcizihg4q58hhh135mvkic'
            onInit={(_evt, editor) => editorRef.current = editor}
            initialValue={currentCapitacion.contenidoCompleto}
            onEditorChange={handleEditorChange}
            init={{
                height: 350,
                menubar: false,
                plugins: [
                    'advlist', 'autolink', 'lists', 'link', 'image',
                    'charmap', 'preview',
                    'anchor', 'searchreplace', 'visualblocks',
                    'code', 'fullscreen',
                    'insertdatetime', 'media', 'table', 'code',
                    'help', 'wordcount'
                ],
                toolbar: 'code | undo redo | blocks | ' +
                    'bold italic forecolor | alignleft aligncenter ' +
                    'alignright alignjustify | bullist numlist
                    outdent indent | ' +
                    'link image media | removeformat | help',
            }}
        />
    </div>

```



```

        // Cargar asignaciones de certificados
        const storedAsignaciones =
JSON.parse(localStorage.getItem('asignacionesCertificados')) || [];
        setAsignacionesCertificados(storedAsignaciones);
    }, []);

    const handleAsignacionChange = (e) => {
        const { name, value } = e.target;
        setAsignacionFormData(prev => ({ ...prev, [name]: value }));
        setFormError('');
    };

    // Función para manejar el envío del formulario de asignación
    const handleAsignacionSubmit = (e) => {
        e.preventDefault();

        // Validaciones
        if (!asignacionFormData.idCapacitacion) {
            setFormError('Por favor, selecciona una capacitación.');
```

return;

```
        }

        // Obtener la capacitación seleccionada para obtener su título
        const capacitacionSeleccionada = capacitaciones.find(
            cap => Number(cap.id) === Number(asignacionFormData.idCapacitacion)
        );

        if (!capacitacionSeleccionada) {
            setFormError('Capacitación no encontrada. Intenta de nuevo.');
```

return;

```
        }

        // Verificar si esta capacitación ya tiene un certificado asignado
        const alreadyAssigned = asignacionesCertificados.some(
            (asignacion) => String(asignacion.idCapacitacion) ===
String(asignacionFormData.idCapacitacion)
        );

        if (alreadyAssigned) {
            setFormError('Esta capacitación ya tiene un certificado asignado.');
```

return;

```
        }

        // Crear el nuevo objeto de asignación
        const nuevaAsignacion = {
            id: uuidv4(), // Generar un ID único para esta asignación
            idCapacitacion: asignacionFormData.idCapacitacion,
            tituloCapacitacion: capacitacionSeleccionada.titulo, // Obtener el título de la
capacitación

```



```

        fechaAsignacion: asignacionFormData.fechaAsignacion,
        urlCertificadoAsociado: CERTIFICADO_GENERICO_URL,
    });

    // Actualizar las capacitaciones para el panel de usuario
    const updatedCapacitacionesForUser = capacitaciones.map(cap => {
        if (Number(cap.id) === Number(asignacionFormData.idCapacitacion)) {
            return {
                ...cap,
                completado: true, // Marcar como completado
                certificadoEmitido: true, // Marcar certificado como emitido
                urlCertificado: CERTIFICADO_GENERICO_URL, // Asignar la URL del
certificado
                fechaEmisionCertificado: asignacionFormData.fechaAsignacion // Usar la
fecha de asignación como fecha de emisión
            };
        }
        return cap;
    });

    setAsignacionesCertificados(prevAsignaciones => [...prevAsignaciones,
nuevaAsignacion]);
    localStorage.setItem('asignacionesCertificados',
JSON.stringify([...asignacionesCertificados, nuevaAsignacion]));

    setCapacitaciones(updatedCapacitacionesForUser);
    localStorage.setItem('capacitacionesData',
JSON.stringify(updatedCapacitacionesForUser));

    // Mostrar mensaje de éxito y cerrar modal
    alert('Certificado asignado y capacitación actualizada exitosamente.');
```

certificado

fecha de asignación como fecha de emisión

```

    setShowAsignarModal(false);
    setAsignacionFormData({
        idCapacitacion: '',
        fechaAsignacion: new Date().toISOString().split('T')[0],
    });
};

// Función para desvincular un certificado de una capacitación
const handleDesvincular = (idAsignacion) => {
    if (window.confirm('¿Estás seguro de que quieres desvincular este certificado de la
capacitación? Esto también lo eliminará de la vista del usuario.')) {
        // Encuentra la asignación que se va a desvincular
        const asignacionToRemove = asignacionesCertificados.find(
            (asignacion) => asignacion.id === idAsignacion
        );

        if (!asignacionToRemove) {
            alert('Asignación no encontrada.');
```

capacitación? Esto también lo eliminará de la vista del usuario.')

asignacion

return;

```

    }

    // 1. Eliminar la asignación de la lista del admin
    const updatedAsignaciones = asignacionesCertificados.filter(
      (asignacion) => asignacion.id !== idAsignacion
    );
    setAsignacionesCertificados(updatedAsignaciones);
    localStorage.setItem('asignacionesCertificados',
JSON.stringify(updatedAsignaciones));

    const updatedCapacitacionesForUser = capacitaciones.map(cap => {
      if (Number(cap.id) === Number(asignacionToRemove.idCapacitacion)) {
        const { completado, certificadoEmitido, urlCertificado,
fechaEmissionCertificado, ...restOfCap } = cap;
        return restOfCap;
      }
      return cap;
    });
    setCapacitaciones(updatedCapacitacionesForUser); // Actualizar el estado de
capacitaciones del admin
    localStorage.setItem('capacitacionesData',
JSON.stringify(updatedCapacitacionesForUser)); // Guardar en localStorage para el usuario

    alert('Certificado desvinculado y capacitación actualizada correctamente.');
```

```

  }
};

// Filtrar capacitaciones que aún no tienen un certificado asignado
const capacitacionesDisponiblesParaAsignar = capacitaciones.filter(cap =>
  !asignacionesCertificados.some(asignacion => String(asignacion.idCapacitacion) ===
String(cap.id))
);

return (
  <div className="certificados-section-wrapper">
    <div className="section-header-with-button">
      <h1>Asignación de Certificados</h1>
      <button
        className="btn-add-new"
        onClick={() => {
          setShowAsignarModal(true);
          setFormError('');
          setAsignacionFormData({
            idCapacitacion: '',
            fechaAsignacion: new Date().toISOString().split('T')[0],
          });
        }}
      >
        + Asignar Certificado
      </button>
    </div>
  </div>
);

```

```

    </div>

    <div className="dashboard-content">
      {asignacionesCertificados.length === 0 ? (
        <p className="empty-state">Aún no has asignado el certificado a ninguna
capacitación.</p>
      ) : (
        <table className="data-table">
          <thead>
            <tr>
              <th>Título de la Capacitación</th>
              <th>Fecha de Asignación</th>
              <th>Acciones</th>
            </tr>
          </thead>
          <tbody>
            {asignacionesCertificados.map((asignacion) => (
              <tr key={asignacion.id}>
                <td>{asignacion.tituloCapacitacion}</td>
                <td>{asignacion.fechaAsignacion}</td>
                <td>
                  <button
                    className="btn-danger"
                    onClick={() => handleDesvincular(asignacion.id)}
                  >
                    Desvincular
                  </button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      )}
    </div>

    {/* Modal para Asignar Certificado */}
    {showAsignarModal && (
      <div className="modal-overlay">
        <div className="modal-content">
          <h2>Asignar Certificado</h2>
          {formError && <p className="error-message">{formError}</p>}
          <form onSubmit={handleAsignacionSubmit}>
            <div className="input-group">
              <label htmlFor="idCapacitacion">Seleccionar
Capacitación:</label>

              <select
                id="idCapacitacion"
                name="idCapacitacion"
                value={asignacionFormData.idCapacitacion}
                onChange={handleAsignacionChange}

```

```

        required
      >
        <option value="" disabled>-- Selecciona una Capacitación
--</option>

        {capacitacionesDisponiblesParaAsignar.map(cap => (
          <option key={cap.id} value={cap.id}>
            {cap.titulo}
          </option>
        ))}
      </select>
    </div>
    <div className="input-group">
      <label htmlFor="fechaAsignacion">Fecha de Asignación:</label>
      <input
        type="date"
        id="fechaAsignacion"
        name="fechaAsignacion"
        value={asignacionFormData.fechaAsignacion}
        onChange={handleAsignacionChange}
        required
      />
    </div>
    <div className="button-group">
      <button type="button" className="btn-cancel" onClick={() =>
setShowAsignarModal(false)}>
        Cancelar
      </button>
      <button type="submit" className="btn-primary">
        Asignar
      </button>
    </div>
  </form>
</div>
</div>
  )}
</div>
);
};

export default Certificados;

```

5. Usuarios

```

import React, { useState, useEffect } from 'react';
import '../CSS/AdminInterface.css';

const Usuarios = () => {
  const [users, setUsers] = useState([]);
  const [showUserForm, setShowUserForm] = useState(false);

```

```

const [currentUser, setCurrentUser] = useState(null);
const [formMessage, setFormMessage] = useState({ text: '', type: '' });
const [formErrors, setFormErrors] = useState({});

// Áreas de trabajo disponibles
const areasDeTrabajo = [
  "Decanato", "Vicedecanato", "Direccion de Carrera", "Docencia", "Investigacion",
  "Secretaria Academica", "Tecnico de Laboratorio", "Soporte Informatico",
  "Administracion de Redes", "Desarrollo de Software Interno",
  "Vinculacion con la Sociedad", "Gestion Administrativa", "Coordinacion de Proyectos",
  "Ciberseguridad", "Inteligencia Artificial", "Bases de Datos", "Desarrollo Web",
  "Robotica", "Analisis de Datos", "Gestion de Calidad", "Asesoría Estudiantil"
];

useEffect(() => {
  try {
    const storedUsers = JSON.parse(localStorage.getItem('users')) || [];
    if (Array.isArray(storedUsers)) {
      setUsers(storedUsers);
    } else {
      console.warn("localStorage 'users' no es un array, se inicializará como vacío.");
      setUsers([]);
    }
  } catch (error) {
    console.error("Error al cargar usuarios de localStorage:", error);
    setUsers([]);
  }
}, []);

const handleChange = (e) => {
  const { name, value } = e.target;
  setCurrentUser(prev => ({ ...prev, [name]: value }));
  setFormErrors(prev => ({ ...prev, [name]: '' }));
  setFormMessage({ text: '', type: '' });
};

const validateCedulaFormat = (cedula) => /^d{10}$/.test(cedula);

const validateUserForm = (userData) => {
  let newErrors = {};
  let isValid = true;

  const currentUsers = users.filter(user => user.cedula !== userData.cedula);

  if (!userData.nombres.trim()) {
    newErrors.nombres = 'Los nombres son obligatorios.'; isValid = false;
  } else if (!/^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$/.test(userData.nombres.trim())) {
    newErrors.nombres = 'Los nombres solo pueden contener letras y espacios.';
    isValid = false;
  }
};

```

```

    }

    const emailRegex = /^[^\s@]+@uleam\.edu\.ec$/;
    if (!userData.correo.trim()) {
        newErrors.correo = 'El correo es obligatorio.'; isValid = false;
    } else if (!emailRegex.test(userData.correo.trim())) {
        newErrors.correo = 'Introduce un correo válido con dominio @uleam.edu.ec.';
        isValid = false;
    } else if (currentUsers.some(user => user.correo === userData.correo.trim())) {
        newErrors.correo = 'Este correo electrónico ya está registrado por otro
        usuario.'; isValid = false;
    }

    if (!userData.telefono.trim()) {
        newErrors.telefono = 'El teléfono es obligatorio.'; isValid = false;
    } else if (!/^\\d{10}$/.test(userData.telefono.trim())) {
        newErrors.telefono = 'El número de teléfono debe contener exactamente 10
        dígitos.'; isValid = false;
    }

    if (!userData.area_trabajo) {
        newErrors.area_trabajo = 'Debes seleccionar un área de trabajo.'; isValid =
        false;
    }

    setFormErrors(newErrors);
    return isValid;
};

const handleUserSubmit = (e) => {
    e.preventDefault();

    if (!currentUser) {
        setFormMessage({ text: 'Error: No hay usuario seleccionado para editar.', type:
        'error' });
        return;
    }

    if (!validateUserForm(currentUser)) {
        setFormMessage({ text: 'Por favor, corrige los errores en el formulario.', type:
        'error' });
        return;
    }

    const updatedUsers = users.map(user =>
        user.cedula === currentUser.cedula ? { ...currentUser, role: user.role } : user
    );

    setUsers(updatedUsers);
    localStorage.setItem('users', JSON.stringify(updatedUsers));

```

```

    setFormMessage({ text: 'Usuario actualizado exitosamente.', type: 'success' });
    setShowUserForm(false);
    setCurrentUser(null);
    setFormErrors({});
  };

  // Abrir el formulario para editar un usuario existente
  const handleEditUser = (user) => {
    setCurrentUser({
      nombres: user.nombres,
      cedula: user.cedula,
      correo: user.correo,
      telefono: user.telefono,
      area_trabajo: user.area_trabajo,
      role: user.role
    });
    setFormErrors({});
    setFormMessage({ text: '', type: '' });
    setShowUserForm(true);
  };

  // Eliminar un usuario
  const handleDeleteUser = (cedula) => {
    if (window.confirm('¿Estás seguro de que deseas eliminar a este usuario?')) {
      const updatedUsers = users.filter(user => user.cedula !== cedula);
      setUsers(updatedUsers); // Actualiza el estado local
      localStorage.setItem('users', JSON.stringify(updatedUsers));

      setFormMessage({ text: 'Usuario eliminado exitosamente.', type: 'success' });
      if (currentUser && currentUser.cedula === cedula) {
        setShowUserForm(false);
        setCurrentUser(null);
      }
    }
  };

  return (
    <div className="usuarios-section-wrapper">
      <h1>Gestión de Usuarios</h1>
      <hr />

      {users.length === 0 && (
        <div className="empty-state-message">
          <p>No hay usuarios registrados.</p>
        </div>
      )}

      {users.length > 0 && (
        <table className="data-table">

```

```

    <thead>
      <tr>
        <th>Nombres</th>
        <th>Cédula</th>
        <th>Correo</th>
        <th>Teléfono</th>
        <th>Área de Trabajo</th>
        <th>Rol</th>
        <th>Acciones</th>
      </tr>
    </thead>
    <tbody>
      {users.map(user => (
        <tr key={user.cedula}>
          <td>{user.nombres}</td>
          <td>{user.cedula}</td>
          <td>{user.correo}</td>
          <td>{user.telefono}</td>
          <td>{user.area_trabajo}</td>
          <td>{user.rol}</td>
          <td className="action-buttons">
            <button className="edit-btn" onClick={() =>
handleEditUser(user)}>Editar</button>
            {user.rol !== 'admin' && (
              <button className="delete-btn" onClick={() =>
handleDeleteUser(user.cedula)}>Eliminar</button>
            )}
          </td>
        </tr>
      )})}
    </tbody>
  </table>
)}

{showUserForm && currentUser && (
  <div className="form-modal-overlay">
    <div className="form-modal-content">
      <h2>Editar Usuario</h2>
      {formMessage.text && (
        <div className={`message ${formMessage.type}`}>
          {formMessage.text}
        </div>
      )}
      <form onSubmit={handleUserSubmit}>
        <div className="form-group">
          <label htmlFor="nombres">Nombres:</label>
          <input
            type="text" id="nombres" name="nombres"
            value={currentUser.nombres} onChange={handleChange}
required

```



```

        />
        {formErrors.nombres && <p className="error-
text">{formErrors.nombres}</p>}}
    </div>
    <div className="form-group">
        <label htmlFor="cedula">Cédula:</label>
        <input
            type="text" id="cedula" name="cedula"
            value={currentUser.cedula} onChange={handleChange}
required
            disabled={true}
        />
    </div>
    <div className="form-group">
        <label htmlFor="correo">Correo:</label>
        <input
            type="email" id="correo" name="correo"
            value={currentUser.correo} onChange={handleChange}
required
        />
        {formErrors.correo && <p className="error-
text">{formErrors.correo}</p>}}
    </div>
    <div className="form-group">
        <label htmlFor="telefono">Teléfono:</label>
        <input
            type="tel" id="telefono" name="telefono"
            value={currentUser.telefono} onChange={handleChange}
required
        />
        {formErrors.telefono && <p className="error-
text">{formErrors.telefono}</p>}}
    </div>
    <div className="form-group">
        <label htmlFor="area_trabajo">Área de trabajo:</label>
        <select
            id="area_trabajo" name="area_trabajo"
            value={currentUser.area_trabajo} onChange={handleChange}
required
        >
            <option value="" disabled>Selecciona una opción</option>
            {areasDeTrabajo.map(area => (
                <option key={area} value={area}>{area}</option>
            ))}
        </select>
        {formErrors.area_trabajo && <p className="error-
text">{formErrors.area_trabajo}</p>}}
    </div>
    <div className="form-actions">
        <button type="submit">Actualizar Usuario</button>

```

```

        <button type="button" onClick={() => {
setShowUserForm(false); setCurrentUser(null); setFormErrors({}); }} className="cancel-btn">
            Cancelar
        </button>
    </div>
</form>
</div>
</div>
    )}
</div>
);
};

export default Usuarios;

```

- AUTH

1. ForgotPassword

```

import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import '../CSS/AuthForms.css';

const ForgotPassword = () => {
    const [cedula, setCedula] = useState('');
    const [email, setEmail] = useState('');
    const [message, setMessage] = useState({ text: '', type: '' });
    const [errors, setErrors] = useState({});

    const navigate = useNavigate();

    const handleChange = (e) => {
        const { name, value } = e.target;
        if (name === 'cedula') {
            setCedula(value);
        } else if (name === 'email') {
            setEmail(value);
        }
        setErrors(prevErrors => ({
            ...prevErrors,
            [name]: ''
        }));
        setMessage({ text: '', type: '' });
    };

    const validateCedulaFormat = (cedulaValue) => {
        // Valida que sean exactamente 10 dígitos numéricos
        return /^\\d{10}$/.test(cedulaValue);
    };
};

```

```

const validateForm = () => {
  let newErrors = {};
  let isValid = true;

  if (!cedula.trim()) {
    newErrors.cedula = 'La cédula es obligatoria.';
    isValid = false;
  } else if (!validateCedulaFormat(cedula.trim())) {
    newErrors.cedula = 'La cédula debe contener exactamente 10 dígitos numéricos.';
    isValid = false;
  }

  if (!email.trim()) {
    newErrors.email = 'El correo es obligatorio.';
    isValid = false;
  } else if (!/^^[^\s@]+@uleam\.edu\.ec$/ .test(email.trim())) {
    newErrors.email = 'Por favor, introduce un correo válido con dominio @uleam.edu.ec.';
    isValid = false;
  }

  setErrors(newErrors);
  return isValid;
};

const handleSubmit = (e) => {
  e.preventDefault();

  if (!validateForm()) {
    setMessage({ text: 'Por favor, corrige los errores en el formulario.', type: 'error'
});
    return;
  }

  // --- CAMBIO CLAVE ---
  const storedUsers = JSON.parse(localStorage.getItem('users') || '[]');

  // Buscar si existe un usuario que coincida con la cédula Y el correo
  const userToReset = storedUsers.find(
    (user) => user.cedula === cedula.trim() && user.correo === email.trim()
  );

  if (userToReset) {
    // Guardar la cédula del usuario a restablecer en localStorage
    // Esto es crucial para que el componente ResetPassword sepa qué usuario actualizar
    localStorage.setItem('userCedulaToReset', cedula.trim());

    setMessage({ text: 'Usuario verificado. Redirigiendo para restablecer contraseña...',
type: 'success' });
  }

```

```

        setTimeout(() => {
            navigate('/reset-password');
        }, 1500);

    } else {
        setMessage({ text: 'Cédula o correo no encontrados en nuestros registros.', type:
'error' });
    }
};

const handleBackToLogin = () => {
    navigate('/login');
};

return (
    <div className="auth-container">
        <div className="auth-header">
            
        </div>

        <div className="auth-content">
            <h2 className="form-title">¿Olvidaste tu Contraseña?</h2>
            <p className="form-description">
                Ingresa tu cédula y correo electrónico registrado para verificar tu cuenta y
                restablecer tu contraseña.
            </p>

            {message.text && (
                <div id="message" className={`message ${message.type}`}>
                    {message.text}
                </div>
            )}

            <form onSubmit={handleSubmit}>
                <div className="input-group">
                    <label htmlFor="cedula">
                        <i className="fas fa-id-card"></i> Cédula:
                    </label>
                    <input
                        type="text"
                        id="cedula"
                        name="cedula"
                        placeholder="Ingresa su número de cédula"
                        value={cedula}
                        onChange={handleChange}
                        required
                    />
                    {errors.cedula && <p className="error-text">{errors.cedula}</p>}
                </div>
            </form>
        </div>
    </div>

```

```

    <div className="input-group">
      <label htmlFor="email">
        <i className="fas fa-envelope"></i> Correo:
      </label>
      <input
        type="email"
        id="email"
        name="email"
        placeholder="ejemplo@uleam.edu.ec"
        value={email}
        onChange={handleChange}
        required
      />
      {errors.email && <p className="error-text">{errors.email}</p>}
    </div>

    <div className="button-group">
      <button type="button" className="btn btn-cancel" onClick={handleBackToLogin}>
        Cancelar
      </button>
      <button type="submit" className="btn btn-primary">
        Verificar y Restablecer
      </button>
    </div>
  </form>
</div>

<footer className="site-footer">
  <p>&copy; 2025 Departamento de Talento Humano ULEAM.</p>
  <p>Contacto: departamentotalentohumanouleam@gmail.com</p>
</footer>
</div>
);
};

export default ForgotPassword;

```

2. Login

```

import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import '../CSS/AuthForms.css';

const ADMIN_AREA = 'Departamento de Talento Humano';

const Login = () => {
  const [formData, setFormData] = useState({
    cedula: '',
    contrasena: ''
  });

```

```

});
const [message, setMessage] = useState({ text: '', type: '' });
const [errors, setErrors] = useState({});

const navigate = useNavigate();

const handleChange = (e) => {
  const { name, value } = e.target;
  setFormData(prevFormData => ({
    ...prevFormData,
    [name]: value
  }));
  setErrors(prevErrors => ({
    ...prevErrors,
    [name]: ''
  }));
  setMessage({ text: '', type: '' });
};

const validateForm = () => {
  let newErrors = {};
  let isValid = true;

  if (!formData.cedula.trim()) {
    newErrors.cedula = 'La cédula es obligatoria.';
    isValid = false;
  } else if (!/^d{10}$/.test(formData.cedula.trim())) {
    newErrors.cedula = 'La cédula debe contener exactamente 10 dígitos numéricos.';
    isValid = false;
  }

  if (!formData.contrasena) {
    newErrors.contrasena = 'La contraseña es obligatoria.';
    isValid = false;
  }

  setErrors(newErrors);
  return isValid;
};

const handleSubmit = (e) => {
  e.preventDefault();

  if (!validateForm()) {
    setMessage({ text: 'Por favor, corrige los errores en el formulario.', type:
'error' });
    return;
  }

  const enteredCedula = formData.cedula.trim();

```

```

const enteredContrasena = formData.contrasena;
let authenticatedUser = null;

// 1. Intentar autenticar como el administrador predefinido
const adminCedula = '1234567890';
const adminContrasena = 'admin123';

if (enteredCedula === adminCedula && enteredContrasena === adminContrasena) {
  authenticatedUser = {
    cedula: adminCedula,
    contrasena: adminContrasena,
    nombres: 'Admin Talento Humano',
    area_trabajo: ADMIN_AREA,
    role: 'admin'
  };
} else {
  // 2. Si no es el admin, buscar en la lista de usuarios registrados
  try {
    // Leer el array de usuarios desde localStorage
    const storedUsers = JSON.parse(localStorage.getItem('users')) || [];
    console.log("Usuarios cargados desde localStorage para autenticación:",
storedUsers);

    // Buscar el usuario en el array
    authenticatedUser = storedUsers.find(
      user => user.cedula === enteredCedula && user.contrasena ===
enteredContrasena
    );

  } catch (error) {
    console.error("Error al leer usuarios de localStorage durante el login:",
error);

    setMessage({ text: 'Ocurrió un error al cargar datos de usuario. Intenta de
nuevo.', type: 'error' });
    return;
  }
}

if (authenticatedUser) {
  setMessage({ text: `¡Inicio de sesión con éxito!`, type: 'success' });
  console.log('Usuario autenticado:', authenticatedUser.cedula, 'Área:',
authenticatedUser.area_trabajo, 'Rol:', authenticatedUser.role);

  localStorage.setItem('currentUserCedula', authenticatedUser.cedula);
  localStorage.setItem('currentUserNombre', authenticatedUser.nombres ||
authenticatedUser.cedula);
  localStorage.setItem('currentUserArea', authenticatedUser.area_trabajo);
  localStorage.setItem('currentUserRole', authenticatedUser.role || 'user');
  localStorage.setItem('currentUser', JSON.stringify(authenticatedUser));
  localStorage.removeItem('registeredUser');
}

```

```

        setTimeout(() => {
            setFormData({ cedula: '', contrasena: '' });
            setMessage({ text: '', type: '' });

            // Redirigir según el rol
            if (authenticatedUser.role === 'admin') {
                navigate('/dashboard');
            } else {
                navigate('/user-dashboard');
            }
        }, 1500);

    } else {
        setMessage({ text: 'Cédula o contraseña incorrectas.', type: 'error' });
    }
};

return (
    <div className="auth-container">
        <div className="auth-header">
            
        </div>

        <div className="auth-content">
            <h2 className="form-title">Iniciar Sesión</h2>

            {message.text && (
                <div id="message" className={`message ${message.type}`}>
                    {message.text}
                </div>
            )}

            <form onSubmit={handleSubmit}>
                <div className="input-group">
                    <label htmlFor="cedula">
                        <i className="fas fa-id-card"></i> Cédula:
                    </label>
                    <input
                        type="text"
                        id="cedula"
                        name="cedula"
                        placeholder="Ingrese su cédula"
                        value={formData.cedula}
                        onChange={handleChange}
                        required
                    />
                    {errors.cedula && <p className="error-text">{errors.cedula}</p>}
                </div>
            </form>
        </div>
    </div>
);

```



```

        <div className="input-group">
          <label htmlFor="contrasena">
            <i className="fas fa-lock"></i> Contraseña:
          </label>
          <input
            type="password"
            id="contrasena"
            name="contrasena"
            placeholder="Ingrese su contraseña"
            value={formData.contrasena}
            onChange={handleChange}
            required
          />
          {errors.contrasena && <p className="error-
text">{errors.contrasena}</p>}
        </div>

        <Link to="/forgot-password" className="link-forgot-password">¿Olvidaste
tu contraseña?</Link>

        <div className="button-group">
          <button type="submit" className="btn btn-primary">
            Entrar
          </button>
        </div>

        <p className="auth-footer-links">
          ¿No tienes una cuenta? <Link to="/register" className="link-
register">Regístrate aquí</Link>
        </p>
      </form>
    </div>

    <footer className="site-footer">
      <p>&copy; 2025 Departamento de Talento Humano ULEAM.</p>
      <p>Contacto: departamentotalentohumanouleam@gmail.com</p>
    </footer>
  </div>
);
};

export default Login;

```

3. Register

```

import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import '../CSS/AuthForms.css';

```

```

const Register = () => {
  // Estado para todos los campos del formulario
  const [formData, setFormData] = useState({
    nombres: '',
    cedula: '',
    correo: '',
    telefono: '',
    contrasena: '',
    confirmar_contrasena: '',
    area_trabajo: ''
  });

  // Estado para el mensaje general de éxito/error
  const [message, setMessage] = useState({ text: '', type: '' });

  // Estado para errores por campo
  const [errors, setErrors] = useState({});

  // Hook para la navegación programática
  const navigate = useNavigate();

  // Maneja los cambios en los inputs y select
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData(prevFormData => ({
      ...prevFormData,
      [name]: value
    }));

    // Limpiar el error específico del campo y el mensaje general cuando el usuario empieza a
    escribir
    setErrors(prevErrors => ({
      ...prevErrors,
      [name]: '' // Limpia el error de este campo
    }));
    setMessage({ text: '', type: '' }); // Limpia el mensaje general
  };

  const validateCedulaFormat = (cedula) => {
    return /^d{10}$/.test(cedula);
  };

  const validateForm = () => {
    let newErrors = {};
    let isValid = true;

    // --- Cargar usuarios existentes para la validación de unicidad ---
    const users = JSON.parse(localStorage.getItem('users')) || [];

    // 1. Validar Nombres:
    //    - No vacío

```

```

// - Solo letras y espacios
if (!formData.nombres.trim()) {
  newErrors.nombres = 'Los nombres son obligatorios.';
  isValid = false;
} else if (!/^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$/ .test(formData.nombres.trim())) {
  newErrors.nombres = 'Los nombres solo pueden contener letras y espacios.';
  isValid = false;
}

// 2. Validar Cédula:
// - No vacía
// - Exactamente 10 dígitos numéricos
if (!formData.cedula.trim()) {
  newErrors.cedula = 'La cédula es obligatoria.';
  isValid = false;
} else if (!validateCedulaFormat(formData.cedula.trim())) {
  newErrors.cedula = 'La cédula debe contener exactamente 10 dígitos numéricos.';
  isValid = false;
} else if (users.some(user => user.cedula === formData.cedula.trim())) {
  newErrors.cedula = 'Esta cédula ya está registrada. Por favor, usa otra.';
  isValid = false;
}

// 3. Validar Correo:
// - No vacío
// - Formato válido de email con dominio @uleam.edu.ec
const emailRegex = /^[^\s@]+@uleam\.edu\.ec$/; // Solo correos @uleam.edu.ec
if (!formData.correo.trim()) {
  newErrors.correo = 'El correo es obligatorio.';
  isValid = false;
} else if (!emailRegex.test(formData.correo.trim())) {
  newErrors.correo = 'Por favor, introduce un correo válido con dominio @uleam.edu.ec.';
  isValid = false;
} else if (users.some(user => user.correo === formData.correo.trim())) {
  newErrors.correo = 'Este correo electrónico ya está registrado. Por favor, usa
otro.';
  isValid = false;
}

// 4. Validar Teléfono:
// - No vacío
// - Solo 10 dígitos numéricos
if (!formData.telefono.trim()) {
  newErrors.telefono = 'El teléfono es obligatorio.';
  isValid = false;
} else if (!/^\d{10}$/ .test(formData.telefono.trim())) {
  newErrors.telefono = 'El número de teléfono debe contener exactamente 10 dígitos
numéricos.';
  isValid = false;
}

```

```

// 5. Validar Contraseña:
//   - No vacía
//   - Mínimo 6 caracteres
if (!formData.contrasena) {
  newErrors.contrasena = 'La contraseña es obligatoria.';
  isValid = false;
} else if (formData.contrasena.length < 6) {
  newErrors.contrasena = 'La contraseña debe tener al menos 6 caracteres.';
  isValid = false;
}

// 6. Validar Confirmar Contraseña:
//   - No vacía
//   - Debe coincidir con la contraseña
if (!formData.confirmar_contrasena) {
  newErrors.confirmar_contrasena = 'Confirmar contraseña es obligatorio.';
  isValid = false;
} else if (formData.contrasena !== formData.confirmar_contrasena) {
  newErrors.confirmar_contrasena = 'Las contraseñas no coinciden.';
  isValid = false;
}

// 7. Validar Área de Trabajo:
//   - No puede ser la opción por defecto (vacía)
if (!formData.area_trabajo) {
  newErrors.area_trabajo = 'Debes seleccionar un área de trabajo.';
  isValid = false;
}

setErrors(newErrors);
return isValid;
});

const handleSubmit = (e) => {
  e.preventDefault();

  // Ejecutar todas las validaciones antes de intentar enviar los datos
  if (!validateForm()) {
    setMessage({ text: 'Por favor, corrige los errores en el formulario.', type: 'error'
  });

  return;
}

// Si todas las validaciones pasan, procede con la lógica de registro
console.log('Datos de registro validados y listos para enviar:', formData);

// --- LÓGICA PARA GUARDAR EL NUEVO USUARIO EN LOCALSTORAGE ---
try {

```

```

    let users = JSON.parse(localStorage.getItem('users')) || []; // Obtener la lista
    actual de usuarios

    // Asignar un rol al nuevo usuario (ej. 'user' por defecto)
    const newUser = { ...formData, role: 'user' };

    users.push(newUser); // Añadir el nuevo usuario a la lista
    localStorage.setItem('users', JSON.stringify(users)); // Guardar la lista actualizada

    setMessage({ text: 'Registro exitoso. ¡Ahora puedes iniciar sesión!', type: 'success'
  });

  setTimeout(() => {
    navigate('/login'); // Redirige al usuario a la página de login
  }, 2000);

} catch (error) {
  console.error("Error al guardar el usuario en localStorage:", error);
  setMessage({ text: 'Ocurrió un error al intentar registrarte. Por favor, intenta de
nuevo.', type: 'error' });
}
};

// Maneja el clic en el botón "Cancelar"
const handleCancel = () => {
  navigate('/login'); // Redirige a la página de login
};

return (
  <>
    <div className="auth-container">
      <div className="auth-header">
        
      </div>

      <div className="auth-content">
        <h1 className="form-title">Registro de Usuario</h1>

        {message.text && (
          <div id="message" className={`message ${message.type}`}>
            {message.text}
          </div>
        )}

        <form id="registerForm" onSubmit={handleSubmit}>
          <div className="input-group">
            <label htmlFor="nombres">Nombres:</label>
            <input
              type="text"
              id="nombres"

```

```

        name="nombres"
        placeholder="Ingrese sus nombres completos"
        required
        value={formData.nombres}
        onChange={handleChange}
      />
      {errors.nombres && <p className="error-text">{errors.nombres}</p>}
    </div>

    <div className="input-group">
      <label htmlFor="cedula">Cédula:</label>
      <input
        type="text"
        id="cedula"
        name="cedula"
        placeholder="Ingrese su número de cédula"
        required
        value={formData.cedula}
        onChange={handleChange}
      />
      {errors.cedula && <p className="error-text">{errors.cedula}</p>}
    </div>

    <div className="input-group">
      <label htmlFor="correo">Correo:</label>
      <input
        type="email"
        id="correo"
        name="correo"
        placeholder="ejemplo@uleam.edu.ec"
        required
        value={formData.correo}
        onChange={handleChange}
      />
      {errors.correo && <p className="error-text">{errors.correo}</p>}
    </div>

    <div className="input-group">
      <label htmlFor="telefono">Teléfono:</label>
      <input
        type="tel"
        id="telefono"
        name="telefono"
        placeholder="Ingrese su número de teléfono"
        required
        value={formData.telefono}
        onChange={handleChange}
      />
      {errors.telefono && <p className="error-text">{errors.telefono}</p>}
    </div>

```

```

<div className="input-group">
  <label htmlFor="contrasena">Contraseña:</label>
  <input
    type="password"
    id="contrasena"
    name="contrasena"
    placeholder="Cree una contraseña"
    required
    value={formData.contrasena}
    onChange={handleChange}
  />
  {errors.contrasena && <p className="error-text">{errors.contrasena}</p>}
</div>

<div className="input-group">
  <label htmlFor="confirmar_contrasena">Confirmar Contraseña:</label>
  <input
    type="password"
    id="confirmar_contrasena"
    name="confirmar_contrasena"
    placeholder="Confirme su contraseña"
    required
    value={formData.confirmar_contrasena}
    onChange={handleChange}
  />
  {errors.confirmar_contrasena && <p className="error-
text">{errors.confirmar_contrasena}</p>}
</div>

<div className="input-group">
  <label htmlFor="area">Área de trabajo:</label>
  <select
    id="area"
    name="area_trabajo"
    required
    value={formData.area_trabajo}
    onChange={handleChange}
  >
    <option value="" disabled>Selecciona una opción</option>
    <option value="Decanato">Decanato</option>
    <option value="Vicedecanato">Vicedecanato</option>
    <option value="Direccion de Carrera">Dirección de Carrera</option>
    <option value="Docencia">Docencia</option>
    <option value="Investigacion">Investigación</option>
    <option value="Secretaria Academica">Secretaría Académica</option>
    <option value="Tecnico de Laboratorio">Técnico de Laboratorio</option>
    <option value="Soporte Informatico">Soporte Informático</option>
    <option value="Administracion de Redes">Administración de Redes</option>
  </select>

```

```

        <option value="Desarrollo de Software Interno">Desarrollo de Software
Interno</option>
        <option value="Vinculacion con la Sociedad">Vinculación con la
Sociedad</option>
        <option value="Gestion Administrativa">Gestión Administrativa</option>
        <option value="Coordinacion de Proyectos">Coordinación de Proyectos</option>
        <option value="Ciberseguridad">Ciberseguridad</option>
        <option value="Inteligencia Artificial">Inteligencia Artificial</option>
        <option value="Bases de Datos">Bases de Datos</option>
        <option value="Desarrollo Web">Desarrollo Web</option>
        <option value="Robotica">Robótica</option>
        <option value="Analisis de Datos">Análisis de Datos</option>
        <option value="Gestion de Calidad">Gestión de Calidad</option>
        <option value="Asesoria Estudiantil">Asesoría Estudiantil</option>
    </select>
    {errors.area_trabajo} && <p className="error-text">{errors.area_trabajo}</p>
</div>

<div className="button-group">
    <button type="button" className="btn btn-cancel" onClick={handleCancel}>
        Cancelar
    </button>
    <button type="submit" className="btn btn-primary">
        Guardar
    </button>
</div>
</form>
</div>

<footer className="site-footer">
    <p>&copy; 2025 Departamento de Talento Humano ULEAM.</p>
    <p>Contacto: departamentotalentohumanouleam@gmail.com</p>
</footer>
</div>
</>
);
};

export default Register;

```

4. ResetPassword

```

import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import '../CSS/AuthForms.css';

const ResetPassword = () => {
    const [newPassword, setNewPassword] = useState('');
    const [confirmPassword, setConfirmPassword] = useState('');

```



```

const [message, setMessage] = useState({ text: '', type: '' });
const [errors, setErrors] = useState({});
const [userCedulaToReset, setUserCedulaToReset] = useState(null);

const navigate = useNavigate();

useEffect(() => {
  // Al cargar el componente, obtener la cédula del usuario que se verificó
  const cedula = localStorage.getItem('userCedulaToReset');
  if (cedula) {
    setUserCedulaToReset(cedula);
  } else {
    // Si no hay cédula, significa que no pasaron por ForgotPassword
    setMessage({ text: 'Acceso no autorizado. Por favor, verifica tu cuenta primero.',
type: 'error' });
    setTimeout(() => navigate('/forgot-password'), 2000);
  }
}, [navigate]);

const validateForm = () => {
  let newErrors = {};
  let isValid = true;

  if (!newPassword) {
    newErrors.newPassword = 'La nueva contraseña es obligatoria.';
    isValid = false;
  } else if (newPassword.length < 6) {
    newErrors.newPassword = 'La contraseña debe tener al menos 6 caracteres.';
    isValid = false;
  }

  if (newPassword !== confirmPassword) {
    newErrors.confirmPassword = 'Las contraseñas no coinciden.';
    isValid = false;
  }

  setErrors(newErrors);
  return isValid;
};

const handleSubmit = (e) => {
  e.preventDefault();

  if (!userCedulaToReset) {
    setMessage({ text: 'No se ha podido identificar al usuario para restablecer la
contraseña.', type: 'error' });
    return;
  }

  if (!validateForm()) {

```

```

        setMessage({ text: 'Por favor, corrige los errores en el formulario.', type: 'error'
    });

    return;
}

// Obtener todos los usuarios del localStorage
const storedUsers = JSON.parse(localStorage.getItem('users') || '[]');

// Encontrar el índice del usuario que estamos intentando restablecer
const userIndex = storedUsers.findIndex(user => user.cedula === userCedulaToReset);

if (userIndex !== -1) {
    // Si el usuario es encontrado, actualizamos su contraseña
    storedUsers[userIndex].contrasena = newPassword;

    localStorage.setItem('users', JSON.stringify(storedUsers));
    localStorage.removeItem('userCedulaToReset');

    setMessage({ text: 'Contraseña restablecida exitosamente. Redirigiendo a inicio de
sesión...', type: 'success' });
    setTimeout(() => {
        navigate('/login');
    }, 2000);
} else {
    setMessage({ text: 'Error: Usuario no encontrado en los registros para restablecer la
contraseña.', type: 'error' });
}
};

if (!userCedulaToReset && !message.text) {
    return (
        <div className="auth-container">
            <div className="auth-content">
                <p>Cargando...</p>
            </div>
        </div>
    );
}

return (
    <div className="auth-container">
        <div className="auth-header">
            
        </div>
        <div className="auth-content">
            <h2 className="form-title">Restablecer Contraseña</h2>
            <p className="form-description">
                Ingresa tu nueva contraseña.
            </p>

```

```

{message.text && (
  <div id="message" className={`message ${message.type}`}>
    {message.text}
  </div>
)}

<form onSubmit={handleSubmit}>
  <div className="input-group">
    <label htmlFor="newPassword">
      <i className="fas fa-lock"></i> Nueva Contraseña:
    </label>
    <input
      type="password"
      id="newPassword"
      name="newPassword"
      placeholder="Ingrese su nueva contraseña"
      value={newPassword}
      onChange={(e) => {
        setNewPassword(e.target.value);
        setErrors(prev => ({ ...prev, newPassword: '' }));
        setMessage({ text: '', type: '' }); // Limpiar mensaje al escribir
      }}
      required
    />
    {errors.newPassword && <p className="error-text">{errors.newPassword}</p>}
  </div>

  <div className="input-group">
    <label htmlFor="confirmPassword">
      <i className="fas fa-lock"></i> Confirmar Contraseña:
    </label>
    <input
      type="password"
      id="confirmPassword"
      name="confirmPassword"
      placeholder="Confirme su nueva contraseña"
      value={confirmPassword}
      onChange={(e) => {
        setConfirmPassword(e.target.value);
        setErrors(prev => ({ ...prev, confirmPassword: '' }));
        setMessage({ text: '', type: '' }); // Limpiar mensaje al escribir
      }}
      required
    />
    {errors.confirmPassword && <p className="error-
text">{errors.confirmPassword}</p>}
  </div>

  <div className="button-group">
    <button type="submit" className="btn btn-primary">

```

```

        Cambiar Contraseña
      </button>
    </div>
  </form>
</div>
<footer className="site-footer">
  <p>&copy; 2025 Departamento de Talento Humano ULEAM.</p>
  <p>Contacto: departamentotalentohumanouleam@gmail.com</p>
</footer>
</div>
);
};

export default ResetPassword;

```

- USER

1. Capacitacion

```

import React, { useState, useEffect, useCallback } from 'react';
import { useNavigate } from 'react-router-dom';
import '../CSS/UserInterface.css';

const Capacitacion = () => {
  const [capacitaciones, setCapacitaciones] = useState([]);
  const [filtroActivo, setFiltroActivo] = useState('all');
  const [capacitacionDetalle, setCapacitacionDetalle] = useState(null);
  const navigate = useNavigate();

  // Estado para el usuario actual
  const [currentUserCedula, setCurrentUserCedula] = useState(null);

  // Función para obtener las inscripciones de un usuario específico
  const getInscripcionesUsuario = useCallback((cedula) => {
    try {
      const inscripcionesDataString = localStorage.getItem('inscripcionesPorUsuario');
      const inscripcionesPorUsuario = inscripcionesDataString ?
JSON.parse(inscripcionesDataString) : {};
      return inscripcionesPorUsuario[cedula] || [];
    } catch (error) {
      console.error("Error al leer inscripcionesPorUsuario de localStorage:", error);
      return [];
    }
  }, []);

  // Función para guardar las inscripciones de un usuario específico
  const saveInscripcionesUsuario = useCallback((cedula, inscripcionesList) => {
    try {
      const inscripcionesDataString = localStorage.getItem('inscripcionesPorUsuario');

```

```

        const inscripcionesPorUsuario = inscripcionesDataString ?
JSON.parse(inscripcionesDataString) : {};
        inscripcionesPorUsuario[cedula] = inscripcionesList;
        localStorage.setItem('inscripcionesPorUsuario',
JSON.stringify(inscripcionesPorUsuario));
    } catch (error) {
        console.error("Error al guardar inscripcionesPorUsuario en localStorage:",
error);
    }
}, []);

useEffect(() => {
    // 1. Obtener la cédula del usuario actual
    const userString = localStorage.getItem('currentUser');
    const user = userString ? JSON.parse(userString) : null;
    const cedula = user ? user.cedula : null;
    setCurrentUserCedula(cedula);

    // 2. Cargar todas las capacitaciones disponibles
    try {
        const storedCapacitaciones =
JSON.parse(localStorage.getItem('capacitacionesData') || '[]');

        // 3. Obtener las inscripciones para el usuario actual
        const userInscripciones = cedula ? getInscripcionesUsuario(cedula) : [];
        console.log(`DEBUG: Inscripciones para usuario ${cedula}:`, userInscripciones);
// DEBUG

        // 4. Mapear las capacitaciones y determinar su estado 'inscrito' basado en el
usuario actual
        setCapacitaciones(storedCapacitaciones.map(cap => {
            const tipoNormalizado = cap.tipoInscripcion
                ? (cap.tipoInscripcion.toLowerCase() === 'libre' ? 'opcional' :
cap.tipoInscripcion.toLowerCase())
                : 'opcional';

            // Determina si esta capacitación está en la lista de inscripciones del
usuario actual
            const isCurrentlyInscrito = tipoNormalizado === 'obligatoria' ||
userInscripciones.includes(cap.id);

            return {
                ...cap,
                inscrito: isCurrentlyInscrito,
                tipoInscripcion: tipoNormalizado
            };
        }));
    } catch (error) {
        console.error("Error al leer o parsear capacitaciones o inscripciones desde
localStorage:", error);
    }
}

```

```

        setCapacitaciones([]);
    }
}, [getInscripcionesUsuario]);

const handleInscribirse = (id) => {
    if (!currentUserCedula) {
        alert("Necesitas iniciar sesión para inscribirte en una capacitación.");
        navigate('/login');
        return;
    }

    setCapacitaciones(prevCapacitaciones => {
        const updatedCapacitaciones = prevCapacitaciones.map(cap => {
            if (cap.id === id && cap.tipoInscripcion === 'opcional') {
                return { ...cap, inscrito: !cap.inscrito };
            }
            return cap;
        });

        // Actualiza las inscripciones del usuario actual en localStorage
        const capacitacionToUpdate = updatedCapacitaciones.find(cap => cap.id === id);

        let currentInscripciones = getInscripcionesUsuario(currentUserCedula);

        if (capacitacionToUpdate.inscrito) {
            // Si la capacitación ahora está inscrita, añadir a la lista del usuario si
no está ya
            if (!currentInscripciones.includes(id)) {
                currentInscripciones.push(id);
                saveInscripcionesUsuario(currentUserCedula, currentInscripciones);
                alert(`¡Te has inscrito exitosamente a la capacitación:
${capacitacionToUpdate.titulo}!`);
            }
        } else {
            // Si la capacitación ahora no está inscrita, remover de la lista del usuario
            const index = currentInscripciones.indexOf(id);
            if (index > -1) {
                currentInscripciones.splice(index, 1);
                saveInscripcionesUsuario(currentUserCedula, currentInscripciones);
                alert(`Te has desinscrito de la capacitación:
${capacitacionToUpdate.titulo}.`);
            }
        }

        return updatedCapacitaciones;
    });
};

const handleMostrarDetalles = (capacitacion) => {
    setCapacitacionDetalle(capacitacion);

```

```

    document.body.style.overflow = 'hidden';
  };

  const handleCerrarDetalles = () => {
    setCapacitacionDetalle(null);
    document.body.style.overflow = 'unset';
  };

  const capacitacionesFiltradas = capacitaciones.filter(cap => {
    const tipoNormalizado = cap.tipoInscripcion;

    switch (filtroActivo) {
      case 'all':
        return true;
      case 'inscrito':
        return cap.inscrito;
      case 'obligatorio':
        return tipoNormalizado === 'obligatoria';
      case 'opcional':
        return tipoNormalizado === 'opcional';
      default:
        return false;
    }
  });

  return (
    <main className="main-content-capacitaciones">
      <div className="capacitaciones-header">
        <div className="filter-buttons">
          <button
            className={`filter-button ${filtroActivo === 'all' ? 'active' : ''}`}
            onClick={() => setFiltroActivo('all')}
          >
            Todas
          </button>
          <button
            className={`filter-button ${filtroActivo === 'obligatorio' ? 'active'
: ''}`}
            onClick={() => setFiltroActivo('obligatorio')}
          >
            Obligatorias
          </button>
          <button
            className={`filter-button ${filtroActivo === 'opcional' ? 'active' :
''}`}
            onClick={() => setFiltroActivo('opcional')}
          >
            Opcionales
          </button>
          <button

```

```

        className={`filter-button ${filtroActivo === 'inscrito' ? 'active' :
    ''}`}

        onClick={() => setFiltroActivo('inscrito')}
    >
        Inscritas
    </button>
</div>
</div>

<div className="legend">
    <div className="legend-item">
        <span className="legend-color-opcional"></span> Opcional
    </div>
    <div className="legend-item">
        <span className="legend-color-inscrito"></span> Inscrito
    </div>
</div>

<div className="capacitaciones-list" id="capacitacionesList">
    {capacitacionesFiltradas.length > 0 ? (
        capacitacionesFiltradas.map((cap) => (
            <div
                key={cap.id}
                className={`capacitacion-card
                    ${cap.inscrito ? 'inscrito' : (cap.tipoInscripcion ===
    'obligatoria' ? 'obligatorio' : 'opcional')}
                `}
                data-type={cap.tipoInscripcion}
            >
                <div className="card-details">
                    <h3>{cap.titulo}</h3>
                    <div className="card-meta">
                        <span><i className="far fa-clock"></i>
    {cap.duracion}</span>
                        <span><i className="fas fa-desktop"></i> Virtual</span>
                    </div>
                </div>
                <div className="card-actions">
                    /* Botón de Inscripción/Desinscripción para cursos
    Opcionales */
                    {cap.tipoInscripcion === 'opcional' && !cap.inscrito && (
                        <button
                            className="action-button inscribirme"
                            onClick={() => handleInscribirse(cap.id)}
                        >
                            <i className="fas fa-user-plus"></i> Inscribirme
                        </button>
                    )}
                    {cap.inscrito && (
                        <button

```



```

        className="action-button inscrito-desinscribir"
        // Solo permite desinscribirse si es opcional
        onClick={cap.tipoInscripcion === 'opcional' ? () =>
handleInscribirse(cap.id) : null}

        disabled={cap.tipoInscripcion === 'obligatoria'} //
Deshabilita si es obligatorio
    >
        <i className="fas fa-check-circle"></i> Inscrito
    </button>
    )}

    {/* Lógica para el botón de "Detalles" o "Ir al Curso" */}
    {/* Si NO está inscrito, mostramos "Detalles" */}
    {!cap.inscrito ? (
        <button
            className="action-button details"
            onClick={() => handleMostrarDetalles(cap)}
        >
            <i className="fas fa-info-circle"></i> Detalles
        </button>
    ) : ( // Si SÍ está inscrito
        <button
            className="action-button go-to-course"
            onClick={() => navigate(`/user-
dashboard/curso/${cap.id}`)}
        >
            <i className="fas fa-play-circle"></i> Ir al Curso
        </button>
    )}
    </div>
</div>
))
) : (
    <div className="no-capacitaciones-placeholder">
        <i className="fa-solid fa-graduation-cap"></i>
        <p>No hay capacitaciones disponibles para mostrar según el filtro
actual.</p>
    </div>
    )}
</div>

{/* Modal para Detalles de la Capacitación */}
{capacitacionDetalle && (
    <div className="modal-overlay" onClick={handleCerrarDetalles}>
        <div className="modal-content" onClick={(e) => e.stopPropagation()}>
            <h2>{capacitacionDetalle.titulo}</h2>
            <div className="modal-meta">
                <span><i className="far fa-calendar-alt"></i>
{capacitacionDetalle.fechaInicio}</span>

```

```

                <span><i className="fas fa-calendar-check"></i>
{capitacionDetalle.fechaFin}</span>
            </div>
            <p>{capitacionDetalle.descripcionCorta}</p>
            <button className="modal-close-button"
onClick={handleCerrarDetalles}>Cerrar</button>
        </div>
    </div>
    )}
</main>
);
};

export default Capacitacion;

```

2. PaginaCursoDetalle

```

import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import '../CSS/PaginaCursoDetalle.css';

const PaginaCursoDetalle = () => {
    const { id } = useParams();
    const navigate = useNavigate();
    const [curso, setCurso] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        setLoading(true);
        setError(null);
        try {
            const storedCapitaciones =
JSON.parse(localStorage.getItem('capitacionesData') || '[]');
            const foundCurso = storedCapitaciones.find(cap => String(cap.id) ===
String(id));

            if (foundCurso) {
                setCurso(foundCurso);
            } else {
                setError(`Curso con ID ${id} no encontrado.`);
            }
        } catch (err) {
            console.error("Error al cargar el curso desde localStorage:", err);
            setError("Error al cargar los detalles del curso.");
        } finally {
            setLoading(false);
        }
    }, [id]);

```

```

    if (loading) {
      return <div className="curso-detalle-container"><p>Cargando curso...</p></div>;
    }

    if (error) {
      return (
        <div className="curso-detalle-container">
          <p className="error-message">{error}</p>
          <button onClick={() => navigate('/user-dashboard/capacitacion')}>Volver a
Capacitaciones</button>
        </div>
      );
    }

    if (!curso) {
      return (
        <div className="curso-detalle-container">
          <p>No se pudo cargar el curso. Por favor, inténtelo de nuevo.</p>
          <button onClick={() => navigate('/user-dashboard/capacitacion')}>Volver a
Capacitaciones</button>
        </div>
      );
    }

    // Si el curso se encontró, muestra su contenido
    return (
      <div className="curso-detalle-container">
        <button className="back-button" onClick={() => navigate('/user-
dashboard/capacitacion')}>
          <i className="fas fa-arrow-left"></i> Volver a Capacitaciones
        </button>
        <h1>{curso.titulo}</h1>
        <div className="curso-meta">
          <span><i className="far fa-calendar-alt"></i> {curso.fechaInicio}</span>
          <span><i className="fas fa-calendar-check"></i> {curso.fechaFin}</span>
          <span><i className="far fa-clock"></i> {curso.duracion}</span>
          <span><i className="fas fa-desktop"></i> Virtual</span>
        </div>
        <p className="curso-descripcion-corta">{curso.descripcionCorta}</p>
        <div className="curso-contenido-completo" dangerouslySetInnerHTML={{ __html:
curso.contenidoCompleto }}></div>

      </div>
    );
  };
};

export default PaginaCursoDetalle;

```

3. UserAnuncios

```
import '../CSS/UserInterface.css';

const UserAnuncios = () => {
  const [novedades, setNovedades] = useState([]);

  useEffect(() => {
    try {
      const storedAnnouncements = JSON.parse(localStorage.getItem('anunciosData') || '[]');
      const activeAnnouncements = storedAnnouncements.filter(ann => {
        return true;
      });
      setNovedades(activeAnnouncements);
    } catch (error) {
      console.error("Error al leer o parsear anuncios desde localStorage:", error);
      setNovedades([]);
    }
  }, []);

  return (
    <section className="section-novedades">
      <h2>
        <i className="fa-solid fa-bullhorn"></i>Anuncios y Novedades
      </h2>
      <div className="novedades-container">
        {novedades.length > 0 ? (
          novedades.map((novedad, index) => (
            <div key={index} className="novedad-card">
              <h3>{novedad.titulo}</h3>
              <p dangerouslySetInnerHTML={{ __html: novedad.contenido }}></p>
              {novedad.fechaPublicacion && <span className="announcement-date">{novedad.fechaPublicacion}</span>}
            </div>
          ))
        ) : (
          <div className="no-novedades-placeholder">
            <i className="fa-solid fa-bell-slash"></i>
            <p>No hay anuncios o novedades recientes.</p>
          </div>
        )
      }
    </section>
  );
};
```

```

    })
  </div>
</section>
);
};

export default UserAnuncios;
```

4. UserCertificados

```
import React, { useState, useEffect } from 'react';
import '../CSS/UserInterface.css';

const UserCertificados = () => {
  const [certificados, setCertificados] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    setLoading(true);
    setError(null);

    try {
      // 1. Obtener la cédula del usuario actual
      const currentUserString = localStorage.getItem('currentUser');
      const currentUser = currentUserString ? JSON.parse(currentUserString) : null;

      if (!currentUser || !currentUser.cedula) {
        setError("No se pudo identificar al usuario. Por favor, inicie sesión.");
        setLoading(false);
        return;
      }

      const userCedula = currentUser.cedula;

      // 2. Cargar todas las definiciones de capacitaciones
      const storedCapacitaciones =
JSON.parse(localStorage.getItem('capacitacionesData') || '[]');

      // 3. Cargar las inscripciones del usuario actual
      const inscripcionesPorUsuarioString =
localStorage.getItem('inscripcionesPorUsuario');
      const inscripcionesPorUsuario = inscripcionesPorUsuarioString ?
JSON.parse(inscripcionesPorUsuarioString) : {};
      const userInscripciones = inscripcionesPorUsuario[userCedula] || [];

      console.log("DEBUG Certificados: Capacitaciones globales:",
storedCapacitaciones); // Debug
```

```

        console.log("DEBUG Certificados: Inscripciones del usuario " + userCedula + ":",
userInscripciones); // Debug

        const certificadosDisponibles = storedCapacitaciones.filter(cap => {
            // Primero, verifica si el usuario está inscrito en esta capacitación
            const estaInscrito = userInscripciones.includes(cap.id);

            // Luego, aplica tus criterios existentes para la emisión del certificado
            return estaInscrito && // Asegurarse de que el usuario esté realmente
inscrito

                cap.completado &&
                cap.certificadoEmitido &&
                cap.urlCertificado;
        }).map(cap => ({
            id: cap.id,
            tituloCurso: cap.titulo,
            fechaEmision: cap.fechaEmisionCertificado || 'Fecha no disponible',
            url: cap.urlCertificado,
        }));

        console.log("DEBUG Certificados: Certificados disponibles para " + userCedula +
":", certificadosDisponibles);

        setCertificados(certificadosDisponibles);

    } catch (err) {
        console.error("Error al cargar certificados desde localStorage:", err);
        setError("Error al cargar tus certificados. Por favor, intenta de nuevo.");
        setCertificados([]);
    } finally {
        setLoading(false);
    }
}, []);

if (loading) {
    return (
        <div className="main-content-certificados">
            <div className="certificados-header">
                <h1><i className="fas fa-award"></i>Mis Certificados</h1>
            </div>
            <div className="certificados-list">
                <div className="no-certificados-placeholder">
                    <p>Cargando certificados...</p>
                </div>
            </div>
        </div>
    );
}

if (error) {

```

```

return (
  <div className="main-content-certificados">
    <div className="certificados-header">
      <h1><i className="fas fa-award"></i>Mis Certificados</h1>
    </div>
    <div className="certificados-list">
      <div className="no-certificados-placeholder">
        <p className="error-message">{error}</p>
        <p>Por favor, inténtalo de nuevo más tarde o contacta al soporte.</p>
      </div>
    </div>
  </div>
);
}

return (
  <div className="main-content-certificados"> { /* Contenedor principal de la página de
certificados */}
    <div className="certificados-header">
      <h1><i className="fas fa-award"></i>Mis Certificados</h1>
    </div>

    {certificados.length > 0 ? (
      <div className="certificados-list">
        {certificados.map(certificado => (
          <div key={certificado.id} className="certificado-card">
            <div className="certificate-icon">
              <i className="fas fa-award"></i>
            </div>
            <div className="card-details">
              <h3>{certificado.tituloCurso}</h3>
              <p>Emitido por: Departamento de Talento Humano ULEAM</p>
              <div className="card-meta">
                <span><i className="fas fa-calendar-check"></i> Fecha de
emisión: {certificado.fechaEmision}</span>
              </div>
            </div>
            <div className="card-actions">
              {certificado.url ? (
                <a
                  href={certificado.url}
                  target="_blank"
                  rel="noopener noreferrer"
                  className="action-button download-pdf-btn"
                >
                  <i className="fas fa-external-link-alt"></i> Ver
Certificado
                </a>
              ) : (

```

```

                                <p className="no-url-message">URL del certificado no
disponible.</p>
                                }}
                                </div>
                                </div>
                                )}}
                                </div>
                                ) : (
                                <div className="no-certificados-placeholder">
                                    <i className="fa-solid fa-file-invoice"></i>
                                    <p>Aún no tienes certificados disponibles.</p>
                                </div>
                                )}
                                </div>
                                );
};

export default UserCertificados;

```

5. UserDashboard

```

import React, { useState, useEffect } from 'react';
import { useNavigate, Link, Outlet, useLocation } from 'react-router-dom';
import '../CSS/UserInterface.css';

const UserDashboard = () => {
    const navigate = useNavigate();
    const location = useLocation();

    const [userName, setUserName] = useState('');
    const [userCedula, setUserCedula] = useState('');
    const [isDropdownOpen, setIsDropdownOpen] = useState(false);

    useEffect(() => {
        const storedName = localStorage.getItem('currentUserName');
        const storedCedula = localStorage.getItem('currentUserCedula');
        if (storedName) {
            setUserName(storedName);
        }
        if (storedCedula) {
            setUserCedula(storedCedula);
        }
    });

    const handleClickOutside = (event) => {
        if (isDropdownOpen && !event.target.closest('.header-right')) {
            setIsDropdownOpen(false);
        }
    };
};

```



```

        document.addEventListener('mousedown', handleClickOutside);
        return () => {
            document.removeEventListener('mousedown', handleClickOutside);
        };

    }, [isDropdownOpen]);

    const toggleDropdown = () => {
        setIsDropdownOpen(prev => !prev);
    };

    const handleLogout = (e) => {
        e.preventDefault();
        localStorage.removeItem('currentUserCedula');
        localStorage.removeItem('currentUserEmail');
        localStorage.removeItem('currentUserArea');
        localStorage.removeItem('currentUserRole');
        localStorage.removeItem('currentUser');
        navigate('/login');
    };

    const isActive = (path) => {
        if (path === '/user-dashboard') {
            return location.pathname === '/user-dashboard';
        }
        return location.pathname.startsWith(path);
    };

    return (
        <div className="user-dashboard-container">
            <header className="main-header-top">
                <div className="header-left">
                    
                </div>

                <nav className="main-nav">
                    <ul>
                        <li>
                            <Link to="/user-dashboard" className={`nav-link
${isActive('/user-dashboard')} ? 'active' : ''}`>
                                <i className="fa-solid fa-house"></i>Inicio
                            </Link>
                        </li>
                        <li>
                            <Link to="/user-dashboard/capacitacion" className={`nav-link
${isActive('/user-dashboard/capacitacion')} ? 'active' : ''}`>
                                <i className="fa-solid fa-graduation-cap"></i>Capacitaciones
                            </Link>
                        </li>
                    </ul>
                </nav>
            </header>
        </div>
    );

```

```

        <li>
          <Link to="/user-dashboard/certificados" className={`nav-link
${isActive('/user-dashboard/certificados')} ? 'active' : ''}`>
            <i className="fa-solid fa-certificate"></i>Mis Certificados
          </Link>
        </li>
      </ul>
    </nav>

    <div className="header-right">
      <button className="user-profile-icon" onClick={toggleDropdown}>
        <i className="fa-solid fa-user-circle"></i>
        <span className="user-name">{userName.split(' ')[0] ||
'Usuario'}</span>
      </button>
      <div className={`user-dropdown ${isDropdownOpen ? 'show' : ''}`>
        <ul>
          <li>
            <Link to="/user-dashboard/perfil" onClick={() =>
setIsDropdownOpen(false)}>
              <i className="fa-solid fa-id-card"></i>Mi Perfil
            </Link>
          </li>
          <li>
            <a href="#" onClick={handleLogout}>
              <i className="fa-solid fa-right-from-bracket"></i>Cerrar
Sesión
            </a>
          </li>
        </ul>
      </div>
    </div>
  </div>
</header>

  <main className="main-content-area">
    <Outlet />
  </main>
</div>
);
};

export default UserDashboard;

```

6. UserPerfil

```

import React, { useState, useEffect } from 'react';
import '../CSS/UserInterface.css';

```

```

const UserPerfil = () => {
  // Estado para almacenar la información del usuario
  const [userData, setUserData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    setLoading(true);
    setError(null);

    try {
      const storedUser = localStorage.getItem('currentUser');

      if (storedUser) {
        // Si hay datos, los parsea de JSON a un objeto JavaScript
        const parsedUser = JSON.parse(storedUser);
        setUserData(parsedUser); // Actualiza el estado con los datos del usuario
      } else {
        // Si no hay datos en localStorage, muestra un mensaje de error
        setError("No se encontraron datos de usuario en la sesión. Por favor, inicie sesión.");
      }
    } catch (err) {
      console.error("Error al leer o parsear datos del localStorage en UserPerfil:", err);
      setError("Error al cargar los datos del perfil. El formato de datos guardados es inválido.");
    } finally {
      setLoading(false);
    }
  }, []);

  return (
    <section className="section-perfil">
      <h2><i className="fa-solid fa-id-card"></i>Mi Perfil de Usuario</h2>
      <div className="perfil-info-container">
        /* Muestra un mensaje de carga mientras se obtienen los datos */
        {loading && (
          <p style={{ textAlign: 'center', color: '#007bff' }}>Cargando datos del perfil...</p>
        )}

        /* Muestra un mensaje de error si algo salió mal */
        {error && (
          <p style={{ textAlign: 'center', color: '#dc3545', fontWeight: 'bold' }}>{error}</p>
        )}

        /* Muestra la información del usuario si los datos están disponibles y no hay errores ni carga */

```

```

        {userData && !loading && !error && (
            <>
                <p><strong>Nombre:</strong> {userData.nombres || 'No
especificado'}</p>
                <p><strong>Cédula:</strong> {userData.cedula || 'No
especificado'}</p>
                <p><strong>Área:</strong> {userData.area_trabajo || 'No
especificado'}</p>
                {userData.correo && <p><strong>Email:</strong> {userData.correo}</p>}
                {userData.telefono && <p><strong>Teléfono:</strong>
{userData.telefono}</p>}
            </>
        )}

        {!userData && !loading && !error && (
            <p style={{ textAlign: 'center', color: '#6c757d' }}>No hay datos de
perfil disponibles en esta sesión.</p>
        )}
    </div>
</section>
);
};

export default UserPerfil;

```

- **Components**

1. App

```

import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

// Importa tus componentes de Autenticación
import Login from './components/Auth/Login';
import Register from './components/Auth/Register';
import ForgotPassword from './components/Auth/ForgotPassword';
import ResetPassword from './components/Auth/ResetPassword';

// Importa tus componentes de Administrador
import AdminDashboard from './components/Admin/AdminDashboard';
import CapacitacionesAdmin from './components/Admin/Capacitaciones';
import AnunciosAdmin from './components/Admin/Anuncios';
import CertificadosAdmin from './components/Admin/Certificados';
import UsuariosAdmin from './components/Admin/Usuarios';

// Importa tus componentes de Usuario
import UserDashboard from './components/User/UserDashboard';
import Capacitacion from './components/User/Capacitacion';
import UserAnuncios from './components/User/UserAnuncios';

```

```

import UserCertificados from './components/User/UserCertificados';
import UserPerfil from './components/User/UserPerfil';
import PaginaCursoDetalle from './components/User/PaginaCursoDetalle';

function App() {
  return (
    <Router>
      <Routes>
        { /* --- RUTAS DE AUTENTICACIÓN --- */ }
        <Route path="/" element={<Login />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/forgot-password" element={<ForgotPassword />} />
        <Route path="/reset-password" element={<ResetPassword />} />

        { /* --- RUTAS DEL DASHBOARD DE ADMINISTRADOR --- */ }
        <Route path="/dashboard" element={<AdminDashboard />} />
        <Route path="capacitaciones" element={<CapacitacionesAdmin />} />
        <Route path="anuncios" element={<AnunciosAdmin />} />
        <Route path="certificados" element={<CertificadosAdmin />} />
        <Route path="usuarios" element={<UsuariosAdmin />} />
      </Route>

      { /* --- RUTAS DEL DASHBOARD DE USUARIOS --- */ }
      <Route path="/user-dashboard" element={<UserDashboard />} />
      <Route index element={<UserAnuncios />} />
      <Route path="capacitacion" element={<Capacitacion />} />
      <Route path="certificados" element={<UserCertificados />} />
      <Route path="perfil" element={<UserPerfil />} />
      <Route path="curso/:id" element={<PaginaCursoDetalle />} />
    </Route>

  </Routes>
</Router>
);
}

export default App;

```

2. Main

```

import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,

```

7. Manual de usuario del sistema web.

Este manual de usuario proporciona una guía detallada sobre cómo interactuar con el sistema de gestión de capacitaciones y anuncios de la ULEAM, tanto para el personal administrativo (rol de Administrador) como para los empleados (rol de Usuario).

- **Acceso al Sistema**

Inicio de Sesión:

- Para acceder al sistema, dirígete a la URL de la aplicación.
- Serás redirigido a la página de inicio de sesión.
- Ingresa tu cédula y contraseña en los campos correspondientes.
- Haz clic en el botón "Iniciar Sesión".
- El sistema validará tus credenciales y te dirigirá al panel de control correspondiente a tu rol.

Funcionalidades para el Rol de Usuario (Empleado)

Una vez que inicies sesión como empleado, tendrás acceso a las siguientes secciones:

Inicio

- Al iniciar sesión, los usuarios son dirigidos a la página de inicio, donde pueden ver un resumen o los anuncios más recientes.

Mi Perfil de Usuario

- Esta sección muestra tu información personal registrada en el sistema.
- Para acceder, haz clic en el ícono de perfil o en el enlace "Mi Perfil" en la barra de navegación superior o lateral.
- Verás detalles como: Nombre, Cédula, Área, Email y Teléfono.

Capacitaciones

- En esta sección, puedes consultar el catálogo de capacitaciones disponibles o tu historial de capacitaciones.
- Navega a la sección "Capacitaciones" desde el menú.
- Podrás ver los detalles de cada capacitación, incluyendo título, descripción, fecha y modalidad.
- Tu estado de inscripción a las capacitaciones se reflejará aquí.

Mis Certificados

- Aquí se muestran todos los certificados que has obtenido de las capacitaciones completadas y emitidas por el Departamento de Talento Humano.
- Haz clic en "Mis Certificados" en el menú de navegación.
- Cada certificado se presenta en una tarjeta individual, mostrando el nombre de la capacitación, la entidad emisora (Departamento de Talento Humano ULEAM) y la fecha de emisión.
- Para ver o descargar el certificado (si está disponible), haz clic en el botón "Ver Certificado" o "Descargar PDF" asociado a cada tarjeta.

Anuncios

- Accede a esta sección para ver todos los anuncios y comunicados importantes publicados por el Departamento de Talento Humano.

Cerrar Sesión

- Para salir de tu sesión de forma segura, haz clic en el botón o enlace "Cerrar Sesión" ubicado en el menú de navegación. Esto borrará tus credenciales del navegador.

Funcionalidades para el Rol de Administrador (Departamento de Talento Humano)

Como Administrador, tienes acceso a funcionalidades completas para gestionar el sistema.

Inicio (Dashboard Principal)

- Al iniciar sesión, serás dirigido al dashboard principal, donde podrás ver un resumen de las actividades o la bienvenida al Departamento de Talento Humano.

Gestión de Usuarios

- En esta sección, puedes administrar la información de los empleados de la ULEAM.
- Ver Usuarios: Accede a una lista completa de todos los usuarios registrados.
- Editar Usuario: Modifica la información de un usuario existente.
- Eliminar Usuario: Retira un registro de usuario del sistema.

Gestión de Capacitaciones

- Permite crear, editar y organizar las capacitaciones ofrecidas por el departamento.
- Crear Nueva Capacitación: Define el título, descripción, fechas, modalidad y otros detalles de una nueva capacitación.
- Editar Capacitación: Actualiza la información de capacitaciones ya existentes.
- Eliminar Capacitación: Retira una capacitación del catálogo.

Gestión de Anuncios

- Esta sección te permite crear y mantener los comunicados oficiales para el personal.
- Crear Nuevo Anuncio: Redacta nuevos comunicados utilizando el editor de texto enriquecido (TinyMCE), que permite aplicar formatos, insertar imágenes, etc.
 - *Uso del Editor TinyMCE:* El campo de contenido del anuncio utiliza un editor WYSIWYG. Puedes usar las herramientas de la barra superior (negrita, cursiva, listas, enlaces, imágenes, etc.) para dar formato al contenido.
- Editar Anuncio: Modifica el título o el contenido de los anuncios ya publicados.
- Eliminar Anuncio: Retira anuncios que ya no son relevantes.

Gestión de Certificados

- Permite gestionar los certificados emitidos tras la finalización de las capacitaciones.
- Emitir Certificado: Genera y asocia un certificado a un usuario para una capacitación específica.

Cerrar Sesión

- Para salir de tu sesión como administrador de forma segura, haz clic en el botón o enlace "Cerrar Sesión" ubicado en el menú de navegación.

8. Conclusión del proyecto.

Este sistema web ha digitalizado eficazmente la gestión de capacitaciones y la publicación de anuncios para el Departamento de Talento Humano de la ULEAM, mejorando la eficiencia y la comunicación interna. Desarrollado con React.js y utilizando `localStorage`, la aplicación es responsiva e intuitiva, cumpliendo con los requerimientos esenciales de gestión de contenido y acceso de usuarios. A pesar de usar almacenamiento local, el proyecto demuestra cómo la tecnología puede modernizar procesos clave, sentando una base sólida para futuras mejoras en la gestión del talento humano.

9. Bibliografía.

- **React.js:** <https://react.dev/>
- **TinyMCE:** <https://www.tiny.cloud/>