Universidad Nacional

Escuela de Informática

Ingeniería de Sistemas

I Ciclo-2025

Documentación Técnica del Proyecto I

Sistema de Gestión de Oficinas

Integrantes

María Isabel Morera Murillo

José Andrés Molina Cambronero

Documentación Técnica del Proyecto 1: Sistema de Administración de Oficinas

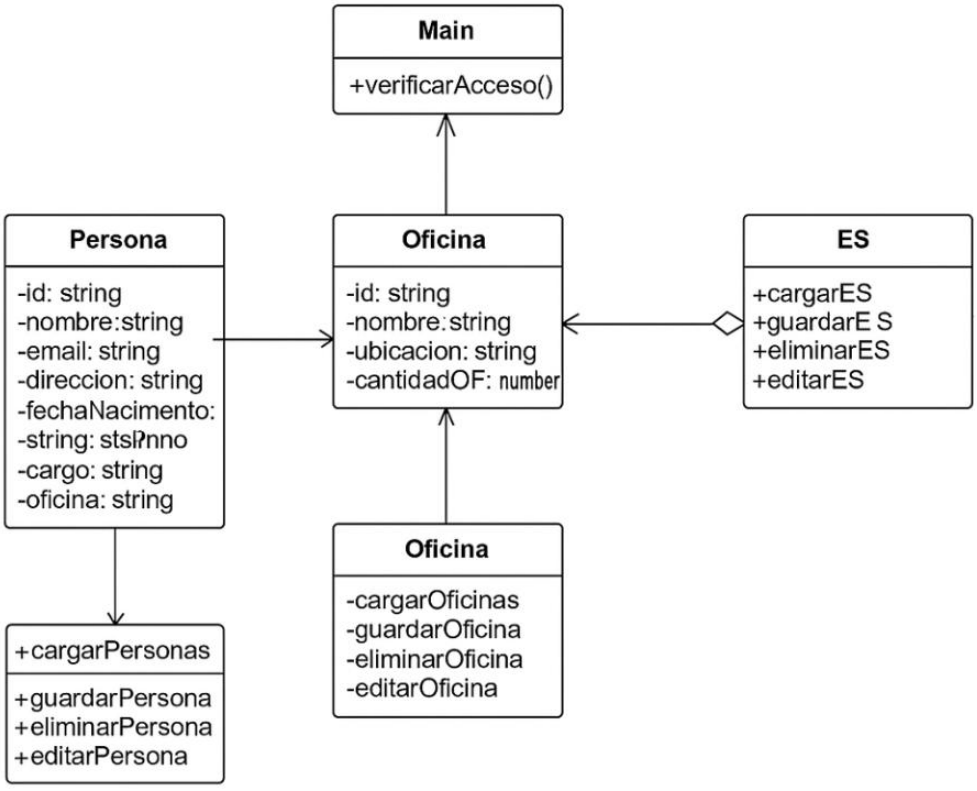
1. Introducción

Este documento tiene como objetivo proporcionar una descripción técnica detallada del sistema de gestión de oficinas y personas. El sistema busca mejorar la administración de estos elementos a través de un conjunto de funcionalidades ampliadas que optimizan la experiencia del usuario, mejoran la seguridad y permiten un control más eficiente de los registros.

El proyecto introduce características como la vinculación entre personas y oficinas, el registro de ingresos y salidas con validaciones de aforo, la implementación de roles de usuario, así como reportes dinámicos para la visualización de datos clave. Además, se ha optimizado la interfaz con Bootstrap 5 y se han agregado herramientas para exportación de datos en formatos PDF y Excel.

Este documento cubre la arquitectura del sistema, la estructura del código, los procedimientos de instalación y configuración, así como las mejores prácticas de mantenimiento y seguridad.

2. Arquitectura del Sistema



2.1 Descripción General

El sistema está desarrollado utilizando HTML, CSS y JavaScript en el frontend, mientras que el backend se simula mediante JavaScript utilizando almacenamiento en `localStorage`. La visualización de datos se apoya en librerías como DataTables y Chart.js para mejorar la interactividad y facilidad de uso.

2.2 Flujo de Datos

1. El usuario ingresa al sistema y accede a los módulos disponibles según su rol.

2. Los registros de personas y oficinas se almacenan en `localStorage`.

3. El usuario puede agregar, editar, eliminar y filtrar registros en tiempo real.

4. Se generan reportes dinámicos con información relevante sobre oficinas y personas.

5. Se validan las entradas y salidas para asegurar la coherencia de los datos.

6. La interfaz proporciona notificaciones y confirmaciones para mejorar la experiencia del usuario.

3. Estructura del Código

El sistema sigue la siguiente estructura para facilitar su mantenimiento:

Archivo css (hojas de estilo): Maneja el apartado visual del sistema

Apariencia de la página:

```css
/* General */
body {
    background-color: dimgray; /* Fondo claro para toda la página */
    font-family: 'Bookman Old Style', sans-serif;/* Fuente de letra */
}

/* Barra de navegación */
.navbar {
    position: sticky;   /* Hace que la barra de navegación sea sticky */
    top: 0;
    background-color: darkred  !important; /* Color de fondo oscuro */
    box-shadow: 0 4px 2px -2px rgba(0, 0, 0, 0.1); /* Sombra suave en el titulo */
    z-index: 1000;
}
```

Archivos js: Organizan la lógica del CRUD correspondiente

Los siguientes archivos manejan el mismo tipo de lógica para guardar agregar y eliminar registros.

- persona.js (módulo de gestión de personas)

Utiliza los siguientes métodos:

cargarPersonas()

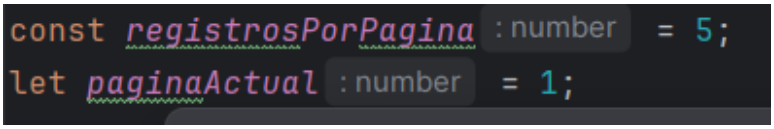
- Carga la lista de personas almacenadas en localStorage y la muestra en una tabla HTML.
- Implementa paginación para mostrar solo un número limitado de registros por página.
- Si no hay datos, muestra un mensaje indicando que no hay información disponible.
- Agrega botones de edición y eliminación a cada fila.

mostrarPaginacion(totalPaginas)

- Genera los botones de paginación según la cantidad de páginas disponibles.
- Permite cambiar de página al hacer clic en un botón de paginación.

- Llama a verificarAcceso() al actualizar la paginación.

Nota: esta función requiere de las siguientes líneas de código incluidas al inicio del persona.js que muestra la constante de registros permitidos por página, así como el número de página de inicio.



eliminarPersona(index)

- Elimina una persona de la lista en localStorage según el índice proporcionado.
- Pide confirmación antes de eliminar el registro.
- Recarga la tabla tras la eliminación y muestra un mensaje de éxito.

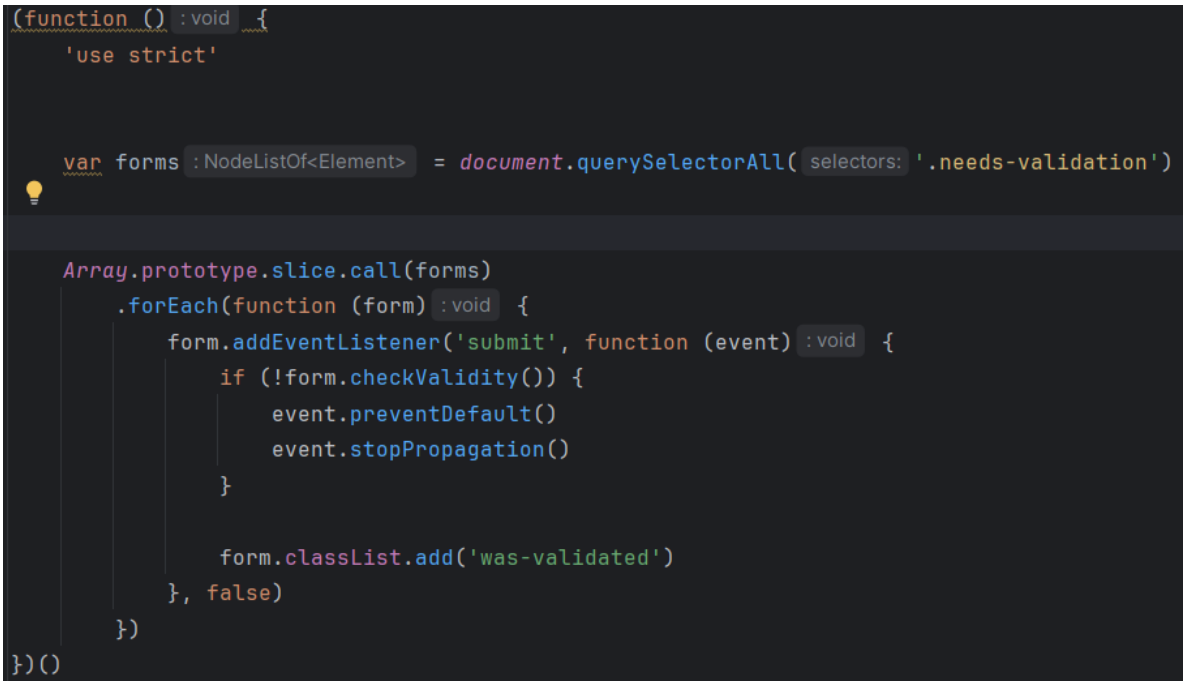
editarPersona(index)

- Guarda en localStorage el índice de la persona a editar.
- Redirige al formulario de edición (form.html).

guardarPersona(event)

- Valida que los campos del formulario no estén vacíos.
- Si se está editando un registro existente, lo actualiza en localStorage.
- Si es un nuevo registro, lo agrega a la lista.
- Muestra mensajes de confirmación y redirige a la página principal.

(function ():void)() (Función de validación)



```
(function () : void {
    'use strict'


    var forms : NodeListOf<Element> = document.querySelectorAll( selectors: '.needs-validation')


    Array.prototype.slice.call(forms)
        .forEach(function (form) : void {
            form.addEventListener('submit', function (event) : void {
                if (!form.checkValidity()) {
                    event.preventDefault()
                    event.stopPropagation()
                }

                form.classList.add('was-validated')
            }, false)
        })
})()
```

- Aplica validaciones de Bootstrap para los formularios.
- Evita el envío de formularios si hay campos inválidos.

filtrarPersonas()

- Filtra la lista de personas según el texto ingresado en un campo de búsqueda.
- Muestra solo las coincidencias dentro de la tabla.
- Si no hay resultados, muestra un mensaje indicando que no se encontraron coincidencias.

exportarPDF()

- Genera un archivo PDF con la lista de personas de la tabla.
- Utiliza jsPDF y autoTable para formatear la información.
- Descarga el archivo como "Lista_de_Personas.pdf".

exportarExcel()

- Convierte la tabla de personas en un archivo Excel.
- Utiliza XLSX.utils.table_to_book para generar el archivo.
- Descarga el archivo como "Lista_de_Personas.xlsx".

oficinas.js (módulo de gestión de oficinas)

cargarOficinas()

- Obtiene la lista de oficinas almacenadas en localStorage y las muestra en una tabla HTML.
- Si no hay datos, muestra un mensaje indicando que no hay oficinas registradas.
- Agrega botones de edición y eliminación a cada fila.

eliminarOficina(index)

- Elimina una oficina de la lista en localStorage según el índice proporcionado.
- Pide confirmación antes de eliminar el registro.
- Recarga la tabla tras la eliminación y muestra un mensaje de éxito.

editarOficina(index)

- Guarda en localStorage el índice de la oficina a editar.
- Redirige al formulario de edición (formOficinas.html).

guardarOficina(event)

- Valida que los campos del formulario no estén vacíos.
- Si se está editando un registro existente, lo actualiza en localStorage.
- Si es un nuevo registro, lo agrega a la lista.
- Muestra mensajes de confirmación y redirige a la página de gestión de oficinas (indexOficina.html).

(function () { ... })() (Función de validación)

- Aplica validaciones de Bootstrap para los formularios.
- Evita el envío de formularios si hay campos inválidos.

initMap()

- Inicializa un mapa de Google Maps con una ubicación por defecto.
- Crea un marcador arrastrable para seleccionar la ubicación de la oficina.
- Si se está editando una oficina, carga su ubicación en el mapa.

actualizarUbicacion(lat, lng)

- Actualiza el campo de ubicación con las coordenadas seleccionadas en el mapa.

ES.js (gestión de ingresos y salidas)

cargarES()

- Obtiene la lista de registros de entradas y salidas almacenados en localStorage y los muestra en una tabla HTML.
- Si no hay datos, muestra un mensaje indicando que no hay registros disponibles.
- Implementa paginación para mostrar un número limitado de registros por página.
- Llama a generarGraficosES() para actualizar los gráficos.

mostrarPaginacionES(totalPaginasES)

- Genera los botones de paginación dinámicamente según la cantidad total de páginas.
- Al hacer clic en un botón, cambia la página y recarga los datos.

eliminarES(index)

- Elimina un registro de entrada o salida de localStorage según su índice.
- Muestra una confirmación antes de eliminar.
- Recarga la tabla después de la eliminación y muestra un mensaje de éxito.

editarES(index)

- Guarda el índice del registro a editar en localStorage y redirige a formES.html.

guardarES(event)

- Guarda o actualiza un registro de entrada o salida en localStorage.
- Valida que todos los campos obligatorios estén llenos.
- Verifica que la oficina seleccionada exista y respeta su capacidad máxima.
- Controla que una persona no pueda registrar una salida sin haber ingresado previamente.
- Almacena el registro y redirige a indexES.html.

exportarPDFES()

- Genera y descarga un archivo PDF con la lista de registros de entrada y salida.

- Usa la biblioteca jsPDF y autoTable para formatear la tabla.

exportarExcelES()

- Genera y descarga un archivo Excel con la lista de registros.
- Usa la biblioteca XLSX para convertir la tabla en un libro de Excel.

generarGraficosES()

- Genera gráficos estadísticos sobre las entradas y salidas de personas y la ocupación de oficinas.
- Usa la biblioteca Chart.js para mostrar gráficos de barras, pastel y dona.
- Calcula cuántas personas han ingresado a cada oficina y cuántas están dentro actualmente.

Evento DOMContentLoaded

- Ejecuta cargarES() y generarGraficosES() al cargar la página para mostrar los datos actualizados.

auth.js (gestión de roles y permisos)

login(event)

- Previene el comportamiento predeterminado del formulario.
- Obtiene el nombre de usuario y la contraseña ingresados.
- Busca en la lista de usuarios predefinidos para validar las credenciales.
- Si el usuario es válido, guarda la autenticación y el rol en localStorage y redirige a main.html.
- Si las credenciales son incorrectas, muestra una alerta de error.

logout()

- Elimina la información de autenticación y rol del localStorage.
- Redirige a la página de inicio de sesión login.html.

verificarAutenticacion()

- Comprueba si el usuario está autenticado revisando localStorage.
- Si no está autenticado, redirige a login.html.

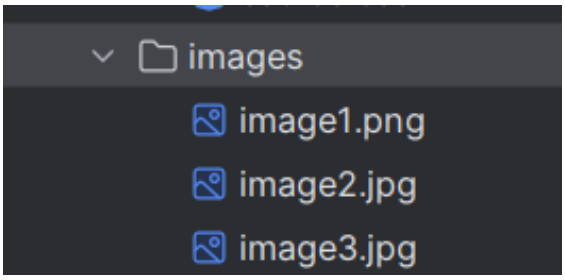
verificarAcceso()

- Obtiene el rol del usuario desde localStorage.
- Si no hay un rol definido, cierra sesión automáticamente.

- Restringe el acceso a elementos de la interfaz según el rol del usuario:
- Los usuarios con rol **"register"** no pueden ver elementos exclusivos de **admin** o **visor**.
- Los usuarios con rol **"visor"** no pueden ver elementos exclusivos de **admin** o **register**.
- Se debe llamar a esta función en main.html para aplicar las restricciones.

Images:

- Contiene recursos visuales (imágenes): image1.png, image2.png, image3.jpeg



Archivos html:

Archivos index: Muestran los datos en tablas.

index.html (CRUD de personas)

Estructura del Documento

- Define un documento HTML con el idioma español.
- Contiene metadatos como el conjunto de caracteres (UTF-8) y el título "Lista de personas".

Enlaces a Hojas de Estilo y Bibliotecas

- Incluye Bootstrap para estilos y componentes responsivos.
- Usa Bootstrap Icons para iconos.
- Carga archivos CSS personalizados desde css/estilos.css.

Scripts y Autenticación

- Importa persona.js para gestionar la lista de personas.
- Carga auth.js y ejecuta verificarAutenticacion() y verificarAcceso() para controlar el acceso de usuarios.

- Integra bibliotecas externas como jspdf y xlsx para exportación de datos.

indexES.html (panel de registro de reportes (entradas y salidas))

Estructura del Documento

- Define un documento HTML con el idioma español.
- Contiene metadatos como el conjunto de caracteres (UTF-8) y el título "Lista de Entradas y Salidas".

Enlaces a Hojas de Estilo y Bibliotecas

- Incluye Bootstrap para estilos y componentes responsivos.
- Usa Bootstrap Icons para iconos.
- Carga archivos CSS personalizados desde css/estilos.css.

Scripts y Autenticación

- Importa ES.js para gestionar las entradas y salidas.
- Carga persona.js para gestionar las personas.
- Carga auth.js y ejecuta verificarAutenticacion() y verificarAcceso() para controlar el acceso de usuarios.
- Integra bibliotecas externas como jspdf, xlsx y chart.js para exportación de datos y generar gráficos.

indexOficina.html (CRUD de oficinas)

Estructura del Documento

- Define un documento HTML con el idioma español.
- Incluye metadatos como el conjunto de caracteres (UTF-8) y el título "Lista de oficinas".
- Se especifica la configuración de la vista para adaptarse a dispositivos móviles (viewport).

Enlaces a Hojas de Estilo y Bibliotecas

- Incluye Bootstrap para estilos y componentes responsivos.
- Usa Bootstrap Icons para iconos.
- Carga archivos CSS personalizados desde css/estilos.css.

Scripts y Autenticación

- Importa oficina.js para gestionar las oficinas.
- Carga auth.js y ejecuta verificarAutenticacion() y verificarAcceso() para controlar el acceso de usuarios.
- Integra la API de Google Maps mediante un script externo.

Archivos form: Formularios para la entrada de datos.

formOficinas.html (formulario de oficinas)

Encabezado:

Título: "Crud de Oficinas"

El formulario tiene un diseño claro y sencillo, utilizando la estructura de Bootstrap para facilitar la creación y la validación de los campos.

Campos del formulario:

Id: Un campo de texto donde se ingresa el identificador único de la oficina.

Nombre: Un campo de texto donde se ingresa el nombre de la oficina.

Ubicación: Un campo oculto (hidden) que se utiliza para almacenar la ubicación de la oficina, con un mapa de Google integrado para seleccionar la ubicación precisa.

Cantidad máxima de personas: Un campo numérico donde se ingresa el número máximo de personas que pueden estar en esa oficina al mismo tiempo.

Botones:

- **Guardar**: Un botón de tipo submit que llama la función guardarOficina(event) para guardar la información de la oficina.
- **Cancelar**: Un botón que redirige al usuario a la página indexOficina.html.

Script de carga y validación:

Carga de datos de oficina: Si se está editando una oficina existente, el formulario carga automáticamente los valores correspondientes de la oficina seleccionada desde localStorage. Los valores son cargados en los campos de entrada.

Autenticación: Se utiliza la función verificarAutenticacion() para asegurar que el usuario esté autenticado antes de acceder a la página.

Google Maps: Se integra un mapa de Google para permitir al usuario seleccionar la ubicación exacta de la oficina. El valor de la ubicación seleccionada se guarda en un campo oculto en el formulario.

Funcionalidad del mapa:

El mapa permite al usuario hacer clic en una ubicación específica, lo que establece la coordenada geográfica de la oficina en el campo oculto ubicacion.

Validación:

Se utiliza la validación de Bootstrap para asegurar que todos los campos sean completados correctamente antes de enviar el formulario. Si un campo no es llenado, se muestra un mensaje de error debajo del campo correspondiente.

Flujo y Funcionalidad:

- **Carga de datos**: Al cargar la página, si existe un índice (editIndex), se cargan los datos de la oficina a editar desde localStorage y se insertan en los campos correspondientes.
- **Guardar**: Al hacer clic en "Guardar", se guarda la nueva oficina o los datos actualizados en localStorage.
- **Mapa**: El mapa permite al usuario seleccionar una ubicación y esa ubicación se guarda como coordenadas geográficas para futuras referencias.

formES.html (formulario de entradas y salidas)

Encabezado:

Título: "Crud de E/S"

El formulario está estructurado de manera limpia y sencilla, con campos de entrada adecuados y validación de Bootstrap para asegurar que los datos ingresados sean correctos.

Campos del formulario:

Persona: Un campo de selección para elegir una persona del listado de personas almacenadas en localStorage. La lista se genera dinámicamente desde los datos almacenados.

Oficina: Un campo de selección para elegir una oficina de una lista de oficinas almacenadas en localStorage. La lista se genera dinámicamente desde los datos almacenados.

Tipo de Registro: Un campo de selección con dos opciones: "Ingreso" y "Salida". Permite al usuario especificar el tipo de registro de la entrada o salida.

Fecha: Un campo de tipo date para ingresar la fecha del evento.

Hora: Un campo de tipo time para ingresar la hora exacta en que se realiza el evento.

Botones:

Guardar: Un botón de tipo submit que llama la función guardarES(event) para guardar los datos del formulario.

Cancelar: Un botón que redirige al usuario a la página indexES.html.

Script de carga y validación:

Carga de personas y oficinas: Al cargar la página, se cargan las personas y oficinas desde localStorage y se insertan dinámicamente en los campos de selección correspondientes.

Edición de registro: Si se está editando un registro existente, los valores del formulario se rellenan con los datos del registro seleccionado, usando el índice editIndex almacenado en localStorage.

Validación: Al igual que en el formulario anterior, se utiliza la validación de Bootstrap para asegurar que todos los campos sean completados correctamente antes de enviar el formulario.

Autenticación y Acceso:

Se verifica que el usuario esté autenticado y tenga los permisos necesarios con las funciones verificarAutenticacion() y verificarAcceso().

Flujo y Funcionalidad:

- **Carga de datos**: Cuando la página se carga, los datos de personas y oficinas se leen desde localStorage y se usan para llenar los campos de selección.
- **Edición de datos**: El formulario es utilizado para editar un registro existente, los valores previos de la entrada o salida seleccionada se cargan en los campos correspondientes.
- **Guardar**: Al hacer clic en "Guardar", se guarda la nueva entrada o salida en localStorage.

form.html (formulario de personas)

Encabezado:

Título: "Crud de personas"

El formulario se encuentra dentro de una estructura de div con un diseño responsivo que adapta los campos a diferentes tamaños de pantalla.

Campos del formulario:

ID: Un campo de texto para ingresar el ID.

Nombre: Campo de texto para ingresar el nombre completo de la persona.

Correo: Campo para ingresar el correo electrónico.

Dirección: Campo de texto para ingresar la dirección de la persona.

Fecha de nacimiento: Campo de tipo date para seleccionar la fecha de nacimiento.

Teléfono: Campo de texto para ingresar el número de teléfono.

Cargo: Campo de texto para ingresar el cargo de la persona.

Estado: Un campo de selección para elegir entre "Activo" o "Inactivo".

Oficina: Un campo de selección donde se elige una oficina de una lista generada dinámicamente a partir de las oficinas disponibles en el localStorage.

Botones:

Guardar: Un botón de tipo submit que llama la función guardarPersona(event) para guardar los datos del formulario.

Cancelar: Un botón que redirige al usuario a la página index.html.

Script de carga y validación:

El formulario incluye un script que carga las oficinas disponibles en el localStorage y muestra las opciones en el campo de selección.

Con editIndex en el localStorage, el formulario carga los datos de la persona para poder editarlos.

Funcionalidades adicionales:

- **Validación**: Se utiliza la validación de Bootstrap para asegurarse de que todos los campos sean completados correctamente antes de enviar el formulario.
- **Autenticación**: Se verifica la autenticación y el acceso del usuario con los scripts auth.js, lo que asegura que solo los usuarios autenticados puedan acceder y modificar la información.

Otros:

main.html (menú principal)

login.html (pantalla de inicio de sesión)

Encabezado:

Título: "Inicio sesión"

El título está centrado en la parte superior de la página.

Formulario:

Campos de entrada:

Username: Campo de texto donde el usuario ingresa su nombre de usuario.

Password: Campo de contraseña donde el usuario debe escribir su contraseña (el texto es ocultado por seguridad).

Íconos: Los campos de entrada para "Username" y "Password" tienen íconos correspondientes de Bootstrap (persona y candado).

Botón de inicio de sesión:

Un botón de tipo submit con un ícono de flecha que se utiliza para enviar el formulario.

Validación de formulario:

Ambos campos (username y password) son obligatorios, como se indica por el atributo required. Si el usuario no llena uno de los campos, el formulario no se enviará y aparecerá un mensaje de error automático.

Scripts y enlaces externos:

Bootstrap: Se incluyen los enlaces a la hoja de estilos de Bootstrap y a los íconos de Bootstrap para darle estilo y funcionalidad al formulario.

JavaScript:

El script auth.js es responsable de la lógica de autenticación. Este script contiene la función login(event) que gestiona el proceso de inicio de sesión.

Flujo de trabajo:

Ingreso de datos:

El usuario ingresa su nombre de usuario y contraseña en los campos correspondientes.

Envío del formulario:

Al hacer clic en el botón "Login", el formulario llama a la función login(event) para procesar la información. Esta función validará el usuario y la contraseña, y redirigirá al usuario a la página principal si la autenticación es correcta.

Estilo:

La página está diseñada con un enfoque simple, clara y profesional, utilizando las herramientas de Bootstrap. El formulario se encuentra centrado en la página con un diseño limpio.

4. Instalación y Configuración

4.1 Requisitos Técnicos

- Navegador compatible con JavaScript (Google Chrome, Firefox, Edge, Safari, Opera, Brave).

- Conexión a Internet para la utilización de la API de Google Maps.

- Configuración de `localStorage` habilitada en el navegador.

4.2 Pasos de Instalación

1. Clonar el repositorio desde GitHub.

2. Abrir `index.html` en un navegador compatible.

3. Configurar los permisos y roles según los usuarios que accederán al sistema.

Registrador de ingresos: Únicamente tiene acceso al registro de entradas y salida.

Para utilizar este usuario debe digitar en el campo correspondiente “register” con la contraseña “7142”.

Administrador: Tiene acceso a todo (registro de personas, oficinas, reportes y estadísticas). Y es capaz de editar cualquier dato.

Para utilizar este usuario debe digitar en el campo correspondiente “admin” con la contraseña “1234”

Visor: Tiene acceso a todo, pero no puede editar, crear o eliminar registros.

Para utilizar este usuario debe digitar en el campo correspondiente “visor” con la contraseña “5678”.

5. Seguridad y Buenas Prácticas

5.1 Validaciones Implementadas

- Restricción de campos obligatorios en formularios.

- Validación de lógica en ingresos y salidas.

- Confirmación antes de eliminar registros.
- Control de aforo en oficinas.

5.2 Medidas de Seguridad

- Implementación de roles y permisos.
- Restricción de accesos según el tipo de usuario.