# Explicación Detallada del Sistema FUNDACITE

Este es un sistema completo para la gestión de solicitudes y asistencias en una organización, con autenticación de usuarios, diferentes niveles de permisos y capacidad de generar reportes en PDF. Vamos a desglosarlo por partes:

## 1. Estructura General

El sistema consta de:

- \*\*Backend\*\*: Aplicación Flask (Python) con base de datos SQLite

- \*\*Frontend\*\*: Páginas HTML con JavaScript para interactividad

- \*\*Base de datos\*\*: Tres modelos principales (Solicitudes, Usuarios, Asistencias)

## 2. Componentes Principales

### 2.1 Backend (app.py)

#### Configuración Inicial

- Usa Flask como framework web

- Configura CORS para permitir peticiones entre dominios

- Establece una clave secreta para sesiones

- Configura la base de datos SQLite usando SQLAlchemy

- Define los departamentos disponibles en la constante `DEPARTAMENTOS`

#### Modelos de Base de Datos

1. \*\*Solicitud\*\*: Almacena todas las tareas/solicitudes con campos como:

- ID, cédula, nombre, dependencia (departamento emisor)

- Tipo, descripción, departamento destino

- Fecha de creación, estado, quién atendió, qué se hizo

2. \*\*User\*\*: Guarda información de usuarios:

- ID, nombre de usuario, hash de contraseña, departamento

3. \*\*Asistencia\*\*: Registra las asistencias del personal:

- ID, nombre, cédula, fecha, hora de entrada, hora editada

#### Rutas y Endpoints

El backend expone varias rutas API:

- \*\*Autenticación\*\*: `/api/login`, `/api/logout`, `/api/current\_user`

- \*\*Solicitudes\*\*: CRUD completo (`/api/solicitudes`)

- \*\*Asistencias\*\*: CRUD completo (`/api/asistencias`)

- \*\*Exportación\*\*: Generación de PDFs (`/api/export-pdf`, `/api/export-asistencias-dia`, `/api/export-asistencias-periodo`)

- \*\*Vistas HTML\*\*: `/`, `/login`, `/asistencias`

### 2.2 Frontend

#### Páginas HTML

1. \*\*login.html\*\*: Formulario de inicio de sesión

2. \*\*index.html\*\*: Vista principal con listado de tareas y creación de nuevas

3. \*\*asistencias.html\*\*: Gestión de registros de asistencia (solo para RRHH)

#### JavaScript

1. \*\*login.js\*\*: Maneja el formulario de login

2. \*\*main.js\*\*: Lógica principal del sistema (gestión de tareas)

3. \*\*asistencias.js\*\*: Lógica específica para gestión de asistencias

## 3. Flujo de Funcionamiento

### 3.1 Autenticación

1. El usuario ingresa credenciales en `/login`

2. El frontend envía las credenciales a `/api/login`

3. El backend verifica y crea una sesión

4. Se redirige a `/` (index.html) si es exitoso

### 3.2 Gestión de Solicitudes

1. \*\*Listado\*\*:

- Se cargan todas las solicitudes del usuario según su departamento

- RRHH ve todas las solicitudes donde es emisor o receptor

- Otros departamentos solo ven las suyas

2. \*\*Creación\*\*:

- Formulario con campos obligatorios

- Validación para evitar duplicados en 30 segundos

- El departamento emisor se establece automáticamente

3. \*\*Modificación\*\*:

- Diferentes permisos según rol:

- Emisor: puede modificar descripción y departamento destino

- Receptor: puede modificar estado, quién atendió y resolución

- RRHH: puede modificar todo

4. \*\*Eliminación\*\*: Solo el emisor puede eliminar solicitudes

### 3.3 Gestión de Asistencias (solo RRHH)

1. \*\*Registro\*\*:

- Formulario simple con nombre, cédula y hora

- La fecha se establece automáticamente

2. \*\*Edición\*\*:

- Permite ajustar la hora de entrada

- Mantiene registro de la hora original y la editada

3. \*\*Visualización\*\*:

- Por día actual, día anterior o fecha específica

- Vista de solo lectura para días pasados

### 3.4 Exportación a PDF

1. \*\*Solicitudes\*\*:

- Filtrado por selección, completadas o pendientes

- Genera un PDF profesional con todos los detalles

2. \*\*Asistencias\*\*:

- Por día específico o por rango de fechas

- Formato tabular claro con agrupación por fecha

## 4. Características Avanzadas

### 4.1 Seguridad y Permisos

- Autenticación por sesiones con Flask

- Contraseñas almacenadas como hash (no texto plano)

- Restricción de acceso a rutas según departamento

- Validación de permisos para cada operación

### 4.2 Experiencia de Usuario

- Notificaciones emergentes para acciones importantes

- Actualización automática de la lista de tareas (polling cada 10s)

- Validación de formularios en cliente y servidor

- Mensajes de error claros

### 4.3 Reportes

- Generación dinámica de PDFs con ReportLab

- Ajuste automático de texto en celdas

- Paginación cuando el contenido excede una página

- Estilos profesionales y consistentes

## 5. Detalles Técnicos Importantes

### 5.1 Base de Datos

- Usa SQLAlchemy como ORM

- Migraciones con Flask-Migrate

- Creación automática de usuarios por defecto al iniciar

### 5.2 Frontend

- Diseño responsive con CSS moderno

- Uso de Font Awesome para íconos

- Manipulación dinámica del DOM

- Comunicación asíncrona con el backend (fetch API)

### 5.3 Backend

- Manejo adecuado de errores y transacciones

- Validación de datos en todas las operaciones

- Separación clara de responsabilidades

- Buenas prácticas de REST para las APIs

## 6. Flujo de Datos

1. \*\*Cliente\*\* realiza acción (ej: crear tarea)

2. \*\*Frontend\*\* valida y envía datos al backend

3. \*\*Backend\*\* procesa, valida y guarda en DB

4. \*\*Backend\*\* responde con resultado

5. \*\*Frontend\*\* actualiza la interfaz según respuesta

6. \*\*Notificación\*\* muestra resultado al usuario

## 7. Consideraciones de Diseño

- \*\*Escalabilidad\*\*: La estructura modular permite agregar fácilmente nuevas funcionalidades

- \*\*Mantenibilidad\*\*: Código bien organizado y comentado

- \*\*Usabilidad\*\*: Interfaz intuitiva con feedback claro

- \*\*Rendimiento\*\*: Uso de polling balanceado con intervalos razonables

Este sistema proporciona una solución completa para la gestión interna de una organización, con especial atención a los diferentes roles y necesidades de cada departamento, todo ello con un enfoque moderno y profesional.

Documentación Técnica del Sistema FUNDACITE

Para Desarrolladores y Mantenedores

Este sistema está diseñado para la gestión interna de solicitudes y asistencias en una organización, con un enfoque modular que permite su expansión y mantenimiento continuo. A continuación, te explico cómo funciona todo el flujo, cómo interactúan sus componentes y cómo puedes seguir desarrollándolo.

1. Arquitectura General

El sistema está dividido en tres capas principales:

Frontend (Interfaz de Usuario)

HTML + CSS + JavaScript:

index.html → Vista principal (gestión de tareas).

asistencias.html → Solo para RRHH (registro de asistencias).

login.html → Autenticación de usuarios.

Lógica en JavaScript:

main.js → Maneja las tareas (crear, editar, eliminar, exportar).

asistencias.js → Lógica de registro y edición de asistencias.

login.js → Procesa el inicio de sesión.

Backend (Servidor Flask - Python)

API RESTful con Flask:

Maneja autenticación (/api/login, /api/logout).

CRUD de tareas (/api/solicitudes).

CRUD de asistencias (/api/asistencias).

Generación de reportes en PDF (/api/export-pdf, /api/export-asistencias-dia).

Base de Datos (SQLite + SQLAlchemy):

Modelos: Solicitud, User, Asistencia.

Base de Datos

SQLite (archivo database.db).

Estructura definida en app.py (Flask-SQLAlchemy).

2. Flujo de Funcionamiento

Autenticación y Roles

Los usuarios se autentican con nombre de usuario y contraseña.

Cada usuario pertenece a un departamento (DTISC, DIAC, RRHH, etc.).

RRHH tiene permisos especiales:

Acceso al módulo de asistencias.

Puede modificar más campos en las tareas.

Gestión de Tareas (Solicitudes)

Listado de Tareas

Se cargan dinámicamente desde /api/solicitudes.

Cada departamento ve solo las tareas que le corresponden (como emisor o receptor).

RRHH ve todas las tareas donde está involucrado.

Creación de Tareas

Formulario con campos obligatorios (cedula, nombre, descripción, departamento\_destino).

El departamento emisor se asigna automáticamente según el usuario logueado.

Edición y Eliminación

Emisor: Puede editar descripción y departamento\_destino.

Receptor: Puede cambiar estado, quien\_atendio y que\_hizo.

RRHH: Puede modificar todo.

Exportación a PDF

Se pueden exportar:

Tareas seleccionadas.

Todas las tareas.

Solo tareas completadas o pendientes.

Gestión de Asistencias (Solo RRHH)

Registro de Asistencias

Formulario simple (nombre, cédula, hora\_entrada).

La fecha se guarda automáticamente.

Edición

Permite corregir la hora de entrada.

Registra tanto la hora original como la hora editada.

Reportes

Se pueden generar PDFs por:

Día específico.

Rango de fechas.

3. Cómo Mantener y Expandir el Sistema

Mantenimiento Básico

Actualizar dependencias:

Flask, SQLAlchemy, ReportLab (para PDFs).

Respaldar la base de datos:

El archivo database.db debe copiarse regularmente.

Monitorear logs:

Flask muestra errores en consola (útil para depuración).

Cómo Añadir Nuevas Funcionalidades

1. Agregar un Nuevo Módulo

Ejemplo: Un módulo de inventario de equipos.

Pasos:

Crear una nueva tabla en la DB (en app.py):

python

class Equipo(db.Model):

id = db.Column(db.Integer, primary\_key=True)

nombre = db.Column(db.String(100))

modelo = db.Column(db.String(50))

departamento = db.Column(db.String(50))

Añadir rutas en Flask (/api/equipos):

python

@app.route('/api/equipos', methods=['GET', 'POST'])

def gestion\_equipos():

if request.method == 'GET':

equipos = Equipo.query.all()

return jsonify([e.to\_dict() for e in equipos])

elif request.method == 'POST':

data = request.get\_json()

nuevo\_equipo = Equipo(nombre=data['nombre'], modelo=data['modelo'], departamento=data['departamento'])

db.session.add(nuevo\_equipo)

db.session.commit()

return jsonify({"message": "Equipo registrado"}), 201

Crear una nueva página HTML (equipos.html) y su JS (equipos.js).

2. Modificar Permisos

Si quieres que DTISC también pueda ver asistencias:

En app.py, cambiar:

python

@app.route('/asistencias')

def asistencias\_page():

if session.get('departamento') not in ['RRHH', 'DTISC']: # ← Añadir DTISC

return redirect('/')

return render\_template('asistencias.html')

3. Mejorar los Reportes PDF

Usar tablas más complejas en ReportLab.

Añadir gráficos con reportlab.graphics.

4. Posibles Mejoras y Expansiones

Seguridad

Implementar JWT en lugar de sesiones Flask.

Añadir captcha en el login.

Usabilidad

Búsqueda avanzada en tareas (por fecha, estado, etc.).

Notificaciones en tiempo real con WebSockets.

Escalabilidad

Migrar a PostgreSQL/MySQL si la base de datos crece.

Usar Celery + Redis para tareas asíncronas (ej: envío de emails).

5. Conclusión

Este sistema está diseñado para ser fácil de mantener y expandir. Si necesitas añadir nuevas funcionalidades:

Define el modelo en app.py.

Crea las rutas API necesarias.

Desarrolla el frontend correspondiente (HTML + JS).

El código está estructurado de forma modular, por lo que puedes añadir componentes sin romper lo existente.