PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

SISTEMAS DE INFORMACIÓN

PLAN DE TRABAJO DE TITULACIÓN

DESARROLLO DE UNA APLICACIÓN WEB PARA EL CONSULTORIO JURÍDICO GRATUITO DE LA PUCE – BACK – END PARÁMETROS

AUTOR

WALTER MATEO BECERRA CALVOPIÑA

DIRECTOR

ING. GUIDO EDUARDO OCHOA MORENO MSC.

QUITO DM, 30 JUNIO DE 2025

Dedicatoria

Este trabajo fue construido, diseñado y culminado con profundo cariño por y para mi familia, por ser el pilar que me ha sostenido en todo momento. A mis padres, por su esfuerzo incansable, el apoyo económico que me brindaron y su amor incondicional. A mis amigos, por motivarme a seguir adelante incluso en los momentos más complejos. Y a Dios, por darme la salud, la paciencia y la oportunidad de culminar esta etapa con gratitud y satisfacción.

Agradecimiento

Agradezco en primer lugar a la **PUCE**, por brindarme una formación académica sólida y por permitirme desarrollar este proyecto en beneficio del Consultorio Jurídico Gratuito de la PUCE. Agradezco especialmente al **Ingeniero Nelson Salgado**, por su guía, sus observaciones constructivas y su apoyo durante todo el proceso. Extiendo también mi gratitud al personal del **Consultorio Jurídico Gratuito**, por compartir su conocimiento y necesidades con claridad, lo que permitió estructurar una solución realista y útil. Finalmente, agradezco a todos quienes, directa o indirectamente, aportaron con su experiencia, motivación y compañía para que este trabajo sea posible.

Resumen

El presente trabajo de titulación consiste en el desarrollo del back-end de una aplicación web para el Consultorio Jurídico Gratuito de la PUCE (CJG), denominado “Balanza Web”, con el objetivo de mejorar la gestión de casos jurídicos y procesos internos. El sistema fue construido bajo una arquitectura modular, empleando tecnologías como Node.js, Express, MySQL y Sequelize, siguiendo el patrón Modelo – Vista - Controlador (MVC). Uno de los módulos centrales desarrollados fue la gestión de parámetros, el cual permite mantener configuraciones reutilizables en distintos formularios del sistema. Se implementaron operaciones crear, editar, eliminar, visualizar (CRUD), validaciones, auditoría automática y pruebas unitarias para garantizar su correcto funcionamiento. Como resultado, se obtuvo una base tecnológica sólida, segura y escalable que mejora significativamente la trazabilidad, eficiencia y control del consultorio.

ABSTRACT

This project presents the development of the backend for a web application called “Balanza Web” designed to support the operations of the Legal Area Office of the Pontificia Universidad Católica del Ecuador. The system was built using Node.js, Express, and MySQL, applying the Model-View-Controller (MVC) pattern to ensure modularity and scalability. A central component of the development was the parameter management module, which allows dynamic control over 36 different catalog tables that support various functional areas of the platform. CRUD operations, logical deletion, and auditing mechanisms were implemented to ensure traceability and data integrity. The project also included unit testing to validate each backend function. The result is a solid backend infrastructure that improves the efficiency, transparency, and maintainability of legal service management within the institutio

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

1. CAPITULO I: Introducción

1.1 JUSTIFICACIÓN

El Consultorio Jurídico Gratuito CJG ha sido un medio importante para los estudiantes de la facultad de Derecho en la PUCE, no solo permite mejorar sus habilidades, sino que también ayuda a personas que no tienen los suficientes recursos económicos para recibir una asistencia y asesoramiento adecuado. Naturalmente, el consultorio jurídico de la PUCE cuenta con una alta demanda de casos y/o situaciones legales que afectan directamente la vida cotidiana de la persona involucrada, en consecuencia, la gestión interna de esta área en particular es un reto que debe ser abordado. En este sentido se comprende la necesidad del desarrollo de una aplicación web para los CJGP con el fin de convertirse en una herramienta de apoyo para los futuros profesionales de esta Facultad, mejorando sus servicios y las necesidades de los usuarios que más lo necesiten.

1.2 PLANTEAMIENTO DEL PROBLEMA

Actualmente, el Consultorio Jurídico Gratuito de la PUCE enfrenta desafíos en la gestión de sus procesos internos, como resultado, la resolución de estos casos se convierte en una carga de tiempo para los estudiantes incluyendo el hecho de que cada caso en particular tiene respectivo nivel de complejidad. Si bien existe un sistema en los CJGP, hay ciertos aspectos que no están contemplados en éste, lo que puede limitar en cierto grado la calidad del servicio, siendo un problema que se complica más debido a la alta demanda de personas en esta área y que buscan soluciones rápidas y efectivas.

En función de esta problemática se plantea la siguiente pregunta principal de investigación:

- ¿Cuál sería la forma más efectiva de desarrollar una aplicación web en el Consultorio Jurídico Gratuito de la PUCE?

Y las siguientes preguntas secundarias:

- ¿Qué características debe tener la aplicación web para asegurar un seguimiento eficiente de los casos?
- ¿Cuáles son las principales necesidades de los usuarios en el consultorio jurídico que la aplicación web debe abordar?
- ¿Qué dificultades enfrentan actualmente los asesores legales al gestionar casos y cómo puede la aplicación web propuesto resolverlas?

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Desarrollar una aplicación web que permita gestionar eficientemente los casos en los consultorios jurídicos.

1.3.2 OBJETIVOS ESPECÍFICOS

- Identificar las necesidades de automatización para facilitar la gestión y seguimiento de casos a los estudiantes.
- Diseñar y codificar métodos y funciones en la parte de backend para la gestión de los casos, incluyendo los elementos para llevar el control y seguimiento de estos.
- Desarrollar la parte backend de la generación de reportes que contemple las particularidades relacionadas a los casos y sus implicaciones.
- Probar las funcionalidades del módulo de primeras consultas y el sistema de autenticación con aplicaciones como Postman o ThunderClient.

1.4 ALCANCE

Este trabajo de titulación tiene como alcance el desarrollo back-end de un sistema para la gestión de parámetros de casos jurídicos en el Consultorio Jurídico gratuito de la PUCE. La implementación abarcará funcionalidades clave, incluyendo:

- Registro de casos: Creación y administración de expedientes jurídicos con información detallada sobre cada proceso legal.
- Registro de actividades de empleados: Gestión de tareas y seguimiento de acciones realizadas por los profesionales del consultorio.
- Reportería: Generación de informes en formatos PDF y Excel, facilitando la consulta y análisis de datos relevantes.
- Parámetros: Gestión de parámetros establecidos y consultados dentro de los Consultorios Jurídicos Gratuitos "CJG". Estos parámetros serán utilizados dentro de todo el sistema y contendrán la validación necesaria de cada caso.

El desarrollo seguirá la metodología de prototipado evolutivo, permitiendo iteraciones y mejoras progresivas basadas en pruebas de usuarios y retroalimentación, brindada por los usuarios. Además, se adoptarán estándares y herramientas tecnológicas definidas por el Centro Informático de la PUCE "Dirección de Informática – Departamento de Desarrollo" para garantizar compatibilidad y calidad en la solución propuesta.

Finalmente, El presente proyecto no se completará en el lapso de tiempo inicialmente establecido, dado el alcance y la complejidad de las funcionalidades previstas. Por esta razón, se espera que futuros estudiantes interesados en el desarrollo de software puedan continuar con el trabajo, finalizando la implementación y optimización de la aplicación web. Esto garantizará que la solución propuesta cumpla plenamente con los objetivos planteados y brinde un aporte significativo al módulo de movilidad humana de los consultorios jurídicos de la Pontificia Universidad Católica del Ecuador.

2 Capítulo II: MARCO TEÓRICO

2.1 APLICACIÓN WEB Y ARQUITECTURA SERVICIOS.

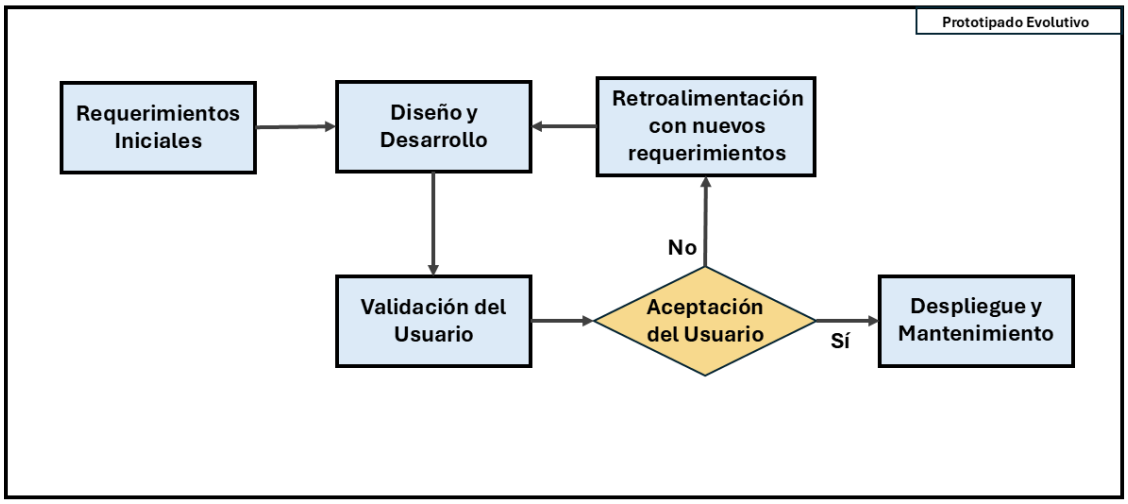
Una aplicación web es un sistema accesible mediante un navegador, sin necesidad de instalar programas en el equipo del usuario. Este tipo de aplicación permite el intercambio de información y funcionalidades desde cualquier ubicación con conexión a internet. En el desarrollo moderno, se ha reemplazado la tradicional arquitectura cliente-servidor por una **arquitectura basada en servicios**, donde los componentes del sistema se comunican a través de interfaces bien definidas, comúnmente utilizando APIs. Esta arquitectura facilita la escalabilidad, el mantenimiento y la reutilización del software, permitiendo que cada servicio funcione de manera independiente pero coordinada. Además, promueve una mejor integración con otras plataformas, ideal para sistemas que requieren modularidad y flexibilidad, como los implementados en entornos jurídicos.

2.2 METODOLOGÍA DE DESARROLLO DE SOFTWARE

La metodología utilizada para el desarrollo de la aplicación web fue "**PROTOTIPADO EVOLUTIVO"**, un enfoque ágil que se centra en la construcción rápida de un prototipo inicial de el sistema web que posteriormente evoluciona a través de múltiples iteraciones. Este prototipo se presenta a los usuarios o stakeholders (abogados de consultorios Jurídicos Gratuitos), quienes aportan retroalimentación valiosa para realizar ajustes, corregir errores y añadir nuevos requerimientos según sus necesidades. De esta manera, el sistema se va refinando progresivamente hasta llegar a una versión final más completa y funcional. Esta metodología es especialmente útil cuando los requerimientos no están completamente definidos desde el inicio o tienden a cambiar con frecuencia, ya que permite adaptarse de forma flexible durante el desarrollo, como es el caso de este proyecto, se realizó un levantamiento preliminar de información que permitió construir la primera versión del

prototipo, facilitando así su validación temprana y una mejor comprensión del producto final deseado. Revisar la **Ilustración 1.**

Ilustración 1 Modelo Prototipado Evolutivo



Nota: Elaboración Propia. Gráfico del proceso de la metodología Prototipado Evolutivo

2.3 BASE DE DATOS

2.3.1 MYSQL

MySQL es un sistema de gestión de bases de datos relacionales de código abierto, no se requieren licencias costosas para usar este motor de BD. Este motor, uno de los más populares del mundo cabe destacar, permite almacenar, organizar y manipular datos de manera eficiente, ya sea para empresas, aplicaciones web o servicios en línea. Las bases de datos actúan como repositorios estructurados donde se guarda información crítica sobre los distintos sistemas. Para interactuar con estas bases, se emplea SQL "Structured Query Language", un lenguaje estándar que facilita operaciones como consultas, actualizaciones, eliminaciones y administración de datos. MySQL opera bajo el modelo cliente-servidor,

donde el servidor aloja los datos y los clientes los solicitan mediante consultas SQL, garantizando un acceso ordenado y seguro a la información.

2.4 LENGUAJE DE PROGRAMACIÓN

2.4.1 JAVASCRIPT

Es un lenguaje de programación de alto nivel para añadir contenido interactivo a páginas web. Este lenguaje es altamente compatible con HTML este lenguaje proporciona una estructura básica a una página y CSS se encarga de colocar estilos al HTML. Este lenguaje de programación se ejecuta directamente en el navegador del usuario en lugar de compilarse previamente, lo que hace que las validaciones se puedan realizar en tiempo real y no solo al recargar una página.

Mientras que otros lenguajes del lado del servidor necesitan recargar la página para procesar datos, JavaScript permite realizar validaciones y actualizaciones en tiempo real. Esta característica lo hace ideal para crear interfaces dinámicas, donde los usuarios pueden interactuar con el contenido sin interrupciones, mejorando significativamente la experiencia de usuario.

2.5 ENTORNO DE DESARROLLO INTEGRADO (IDE)

2.5.1 VISUAL STUDIO CODE

Visual Studio Code es una plataforma diseñada por Microsoft la cual se centra en la codificación, este software es de código abierto y de uso completamente gratuito, existe un IDE con un nombre similar, Visual Studio Comunity, pero este se diferencia por su compatibilidad con tecnologías. Visual Studio Comunity está preparado para ser 100%

compatible con aplicaciones propias de Microsoft mientras que Visual Studio Code es capaz de adaptarse a cualquier lenguaje a partir de sus extensiones hechas por la comunidad.

2.6 FRAMEWORK DE DESARROLLO BACK-END

2.6.1 NODE.JS

Node.js es un entorno de ejecución de un solo hilo en este caso es de código abierto y el funcionamiento multiplataforma facilita la creación de aplicaciones de red y del lado del servidor son más rápidas y escalables. Se ejecuta en el motor de ejecución de JavaScript V8, y utiliza una arquitectura de E/S basada en eventos y sin bloqueos, lo que la hace eficiente y adecuada para aplicaciones en tiempo real. Es un entorno de ejecución que incluye todo lo necesario para ejecutar un programa escrito en JavaScript.

2.7 BIBLIOTECA DE HERRAMIENTAS

2.7.1 JWT (JSON WEB TOKEN)

JWT es un estándar para la creación de tokens para realizar una autenticación apropiada en aplicaciones y también para Web las APIs se encuentran dentro del entorno de JWT, se utiliza un formato JSON para transmitir información de manera confiable. Un JSON Web Token "JWT" está compuesto por tres partes fundamentales:

- El Header o encabezado, que especifica el tipo de token y el algoritmo de firma utilizado.
- El Payload más conocido como cuerpo o carga útil, que contiene la información que se desea transmitir, como la identidad del usuario y sus permisos.
- La Signature o firma, que garantiza la integridad del token, asegurando que no ha sido modificado desde su creación.

Estas tres partes trabajan juntas para proporcionar autenticación y seguridad en aplicaciones y en las APIs.

2.7.2 BCRYPT

Bycript es un método de hash de contraseña derivado del cifrado "Blowfish". La derivación de las claves es más lenta, por lo que se utiliza para dificultar que un atacante encuentre una contraseña. Bycript es un método de contraseña de cifrado creado para salvaguardarlos de intrusiones dañinas. A diferencia de otros métodos más antiguos como MD5, que son muy rápidos, Bcrypt está hecho para ser deliberadamente lento y seguro.

2.7.3 MULTER

Multer es un middleware que se maneja y se encuentra dentro mediante Node.Js para la carga y producción de archivos tipo PDF, JPG y sus variantes, etc. Multer brinda a los desarrolladores la capacidad de especificar restricciones de tamaño de archivo útil para el sistema de CJG, configurar opciones de almacenamiento y aplicar filtros de archivos.

2.8 MODELADO DE DATOS

2.8.1 SEQUELIZE

Sequelize es un mapeo objeto relacional “ORM” utilizado por Node.js que ayuda al uso de base de datos en nuestro trabajo. Este ofrece un sólido soporte para transacciones, garantizando operaciones confiables y atómicas en su base de datos. Al utilizar el modelo Sequelize, se pueden definir e interactuar con sus estructuras de datos de manera eficiente.

Tiene compatibilidad con motores como: PostgreSQL, MySQL, SQLite y Microsoft SQL Server.

2.9 CONTROL DE VERSIONES

2.9.1 IMPORTANCIA DEL CONTROL DE VERSIONES

Al trabajar con control de versiones, implica siempre tener un respaldo de cada versión, esto indica los cambios hechos de una versión a otra de un desarrollo. En caso de que exista un error, se puede regresar a la versión anterior para poder tener la versión estable antes del error y tener un control total con respecto al programa y a su funcionamiento. Al mismo tiempo también un control de versiones nos indica quien hizo el ultimo cambio. Es importante trabajar con control de versiones porque todas las partes interesadas y todos los miembros del equipo pueden trabajar fluidamente en muchos archivos al mismo tiempo. Los sistemas de control de versiones permiten fusionar todas las ediciones y los cambios del código en un repositorio centralizado.

2.9.2 GIT Y GITHUB

Git es un sistema de control de versiones de código abierto. Facilita este tipo de colaboración de proyectos a través del control de versiones distribuido de los archivos que residen en los repositorios tanto como para el front-end como para el back-end. Permite integrar los flujos de trabajo realizados por varios colaboradores a lo largo del tiempo en relación con un repositorio determinado. Y GitHub hospeda estos repositorios Git en la web. Es indispensable conocer que el flujo de trabajo colaborativo se incrementa exponencialmente ya que se trabaja con versiones en tiempo real, hay que tener mucho cuidado con el comando “push” ya que se pueden perder cambios realizados por eso la comunicación es clave.

2.10 BUENAS PRÁCTICAS DE DESARROLLO BACKEND

2.10.1 SEGURIDAD EN APIS

Las aplicaciones web modernas dependen en gran medida de las API para su funcionamiento, a diferencia de los sistemas monolíticos tradicionales que integraban todas sus funciones internamente. Sin embargo, estas interfaces de programación representan un riesgo de seguridad al exponer puntos de acceso externos a la aplicación.

La seguridad de las API consiste precisamente en proteger estos canales de comunicación contra posibles ataques, que suelen manifestarse como explotación de vulnerabilidades, fallos en la autenticación, errores de autorización o avalanchas de solicitudes en períodos cortos, mientras que un sistema sin controles permite solicitudes ilimitadas, una medida efectiva de protección es implementar límites de frecuencia para las peticiones. Cuando un cliente excede el número de solicitudes permitidas en un intervalo de tiempo determinado, el mecanismo de limitación de velocidad puede descartar o bloquear temporalmente sus peticiones, protegiendo así la disponibilidad del servicio y así nosotros podemos ir previniendo posibles ataques de denegación de servicio.

2.10.2 MANEJO DE ERRORES

Las APIs bien gestionadas facilitan el diagnóstico y solución de problemas, a diferencia de sistemas mal implementados que generan discrepancias entre el equipo con respecto a los errores obtenidos. Usar códigos HTTP estandarizados como el Error 404 para recursos inexistentes o el Error 500 para errores internos del servidor y mensajes de error detallados permite identificar rápidamente los fallos, mientras que una API sin pruebas puede fallar en producción, realizar pruebas exhaustivas garantiza su correcto funcionamiento, siendo esta la medida más recomendada para prevenir errores críticos.

2.11 MIDDLEWARE

El middleware como su palabra se traduce es un componente intermedio que actúa como puente entre el sistema, las aplicaciones y otros servicios, permitiendo una comunicación fluida y segura dentro de la arquitectura del software. En el contexto del desarrollo web, los middleware se utilizan principalmente en el back-end para interceptar, procesar y modificar las solicitudes y respuestas que fluyen entre el cliente y el servidor. Estas funciones pueden incluir autenticación de usuarios, registro de actividad, validación de datos, manejo de errores o incluso control de acceso a determinadas rutas. Gracias a su estructura modular, el middleware permite separar responsabilidades, mantener el código organizado y facilitar futuras modificaciones o también se puede dar el caso en futuras ampliaciones del sistema. En tecnologías como Node.js con frameworks como Express, el uso de middleware es fundamental para construir aplicaciones web robustas, seguras y escalables. Dando así un sistema competente ante las necesidades.

3 CAPITULO III: ANÁLISIS Y DISEÑO DE LA APLICACIÓN

3.1 ANÁLISIS DE REQUERIMIENTOS

Para poder identificar los requisitos funcionales y no funcionales y no funcionales de la aplicación web denominada “Balanza Web”, se mantuvieron reuniones entre el equipo desarrollador y los distintos abogados administrativos de CJG para poder entender el flujo principal de la información y deficiencias del actuales del sistema para así poder comprender sus necesidades y prioridades.

Al garantizar, con las distintas reuniones en CJG, un claro entendimiento del flujo principal de información y la estructura organizacional de CJG, se procedió a revisar de manera exhaustiva las funcionalidades que el sistema “Balanza Web” requeriría y se procedió a realizar los casos de uso.

3.1.1 REQUERIMIENTOS FUNCIONALES:

- Gestión de Casos: Se necesita realizar el registro, la edición, y la consulta de los casos que se presentan en CJG. Además de un seguimiento al mismo caso en tiempo real.

- Autenticación y Autorización: Se necesita un registro seguro de usuarios internos.

- Generación de Reportes: Se necesita la generación de un reporte general periódicamente con fechas seleccionables, que contenga toda la información del caso, este debe ser generado en Excel. Debido a que se lo utilizará para otro sistema gubernamental.

- Gestión de parámetros: Se necesita tener el control de los parámetros que el sistema maneja para activar o inhabilitar los distintos campos del sistema.

- Control y Auditoría: Se necesita llevar un registro de todas las acciones (CRUD) realizadas dentro del sistema.

3.1.2 REQUERIMIENTOS NO FUNCIONALES:

- Seguridad: Indispensable el uso de hashing para la seguridad de las contraseñas con Bcrypt.

- Escalabilidad: Se implemento una arquitectura modular dando como resultado una escalabilidad mayor y se potencio con el uso de Node.js y Sequelize ORM para Base de Datos.

- Mantenimiento: El acercamiento y el trabajo en conjunto con la dirección de Informática nos llevo a la implementación de los estándares que maneja la DI - Desarrollo.

- Disponibilidad: El sistema debe estar activo 24/7 de ser posible, y se garantiza debido a que su alojamiento se encuentra en los servidores de la PUCE.

- Compatibilidad: El desarrollo debe estar bajo los lineamientos de la PUCE y seguir los estándares anteriormente mencionados para garantizar la integración del sistema.

- Despliegue: Facilidad de uso debido a la programación de APIs desarrolladas bajo estándares REST que permitan pruebas sencillas mediante el uso de herramientas como Postman o ThunderClient.

3.2 ESPECIFICACIÓN DE REQUERIMIENTOS

3.2.1 CASO DE USO

A continuación, se presentan los diagramas de Caso de Uso enfocados en el módulo de parámetros desarrollados con la herramienta de LucidChart, estos diagramas permiten visualizar de una manera más efectiva el proceso y el flujo que tiene sistema en el módulo de parámetros mediante las interacciones que tiene el usuario con el sistema específicamente con las funcionalidades del backend relacionadas a la gestión de parámetros. Estás funcionalidades incluyen la creación, la edición, y el borrado lógico de un parámetro del sistema con el fin de facilitar la gestión de los parámetros y asegurando los valores que impactarán directamente en el funcionamiento integral del flujo principal del sistema.

3.2.2 DIAGRAMAS CASO DE USO

3.2.2.1 CASO DE USO REQUERIMIENTOS GENERALES.

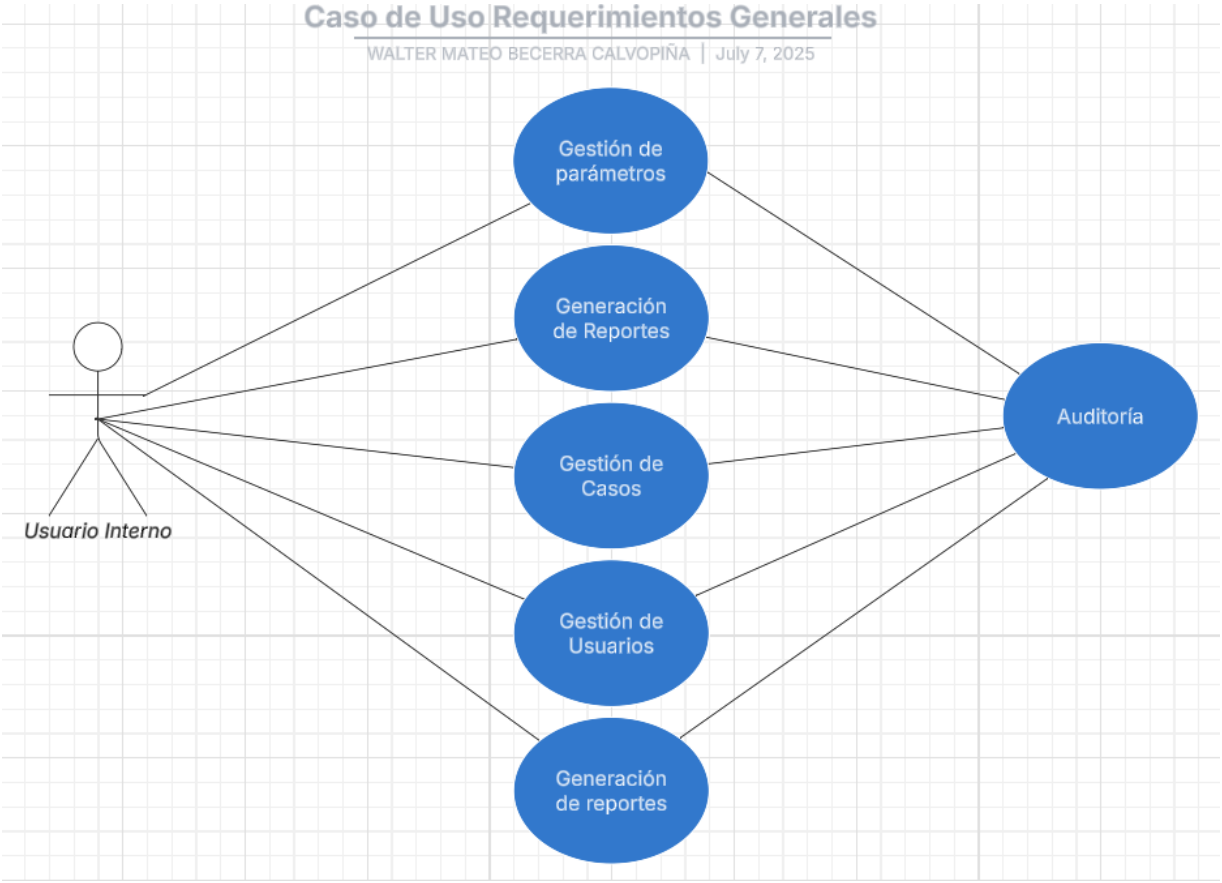
3.2.2.2 ACTOR

- Usuario Interno
    - Rol administrativo
    - Estudiante
    - Abogado
    - Secretaría

3.2.2.3 DESCRIPCIÓN:

El caso de uso de requerimientos generales del sistema web “Balanza Web”, como se muestra en la **Ilustración 2,** está diseñado con todo lo que engloba la aplicación web, el usuario interno es la persona que se encuentra usando la aplicación, estos pueden tener el rol de administrativo, estudiante, abogado, secretaria cada uno de ellos tiene su módulo con sus distintos procesos.

Ilustración 2 Caso de Uso Requerimientos Generales



Nota: Elaboración Propia. Diagrama General de Balanza Web.

3.2.2.2 CASO DE USO GESTIÓN PARÁMETROS

3.2.2.1.1 ACTOR

- Usuario Interno (Rol Administrador)

3.2.2.1.2 DESCRIPCIÓN:

El caso de uso denominado “Gestión de Parámetros” como se muestra en la **Ilustración 3** únicamente lo puede manejar el usuario con rol administrador, le permite administrar las operaciones de crear, editar, actualizar, eliminar, consultar o más conocido como CRUD sobre los parámetros en el sistema web. Este Caso de Uso tiene 4 subdivisiones exclusivas para cada acción del CRUD.

3.2.2.1.3 INGRESAR PARÁMETRO

- **Actor:** Usuario Interno
- **Descripción:** En esta sección se le permite al usuario crear un nuevo parámetro.
- **Flujo Principal:**

1. El usuario interno se dirige a la sección de Gestión de Parámetros.
2. Una vez dentro de la ventana de Gestión de Parámetros, el usuario interno procede a seleccionar en el Combo Box uno de los parámetros del sistema.
3. El sistema le devolverá los parámetros gestionados actualmente, y procederá a seleccionar la opción “Crear”
4. El sistema reflejara un modal solicitando el Nombre del nuevo parámetro.
5. El usuario interno completa el campo y guarda el registro.
6. El sistema verifica la creación del nuevo parámetro y devolverá una notificación de “agregado correctamente”.

3.2.2.1.4 MODIFICAR PARÁMETRO

- **Actor:** Usuario Interno
- **Descripción:** En esta sección se le permite al usuario modificar un parámetro.
- **Flujo Principal:**

1. El usuario interno se dirige a la sección de Gestión de Parámetros.

2. Una vez dentro de la ventana de Gestión de Parámetros, el usuario interno procede a seleccionar en el Combo Box uno de los parámetros del sistema que se desea modificar.
3. Una vez seleccionado el parámetro, se desplegarán los parámetros activos, se selecciona el ícono de editar “lápiz”.
4. El sistema reflejará un modal donde se puede modificar el nombre del parámetro.
5. Se guardan los cambios.
6. El sistema validará los cambios con una notificación “Modificado correctamente”.

3.2.2.1.5 ELIMINAR PARÁMETRO

- **Actor:** Usuario Interno
- **Descripción:** En esta sección se le permite al usuario eliminar un parámetro.
- **Flujo Principal:**

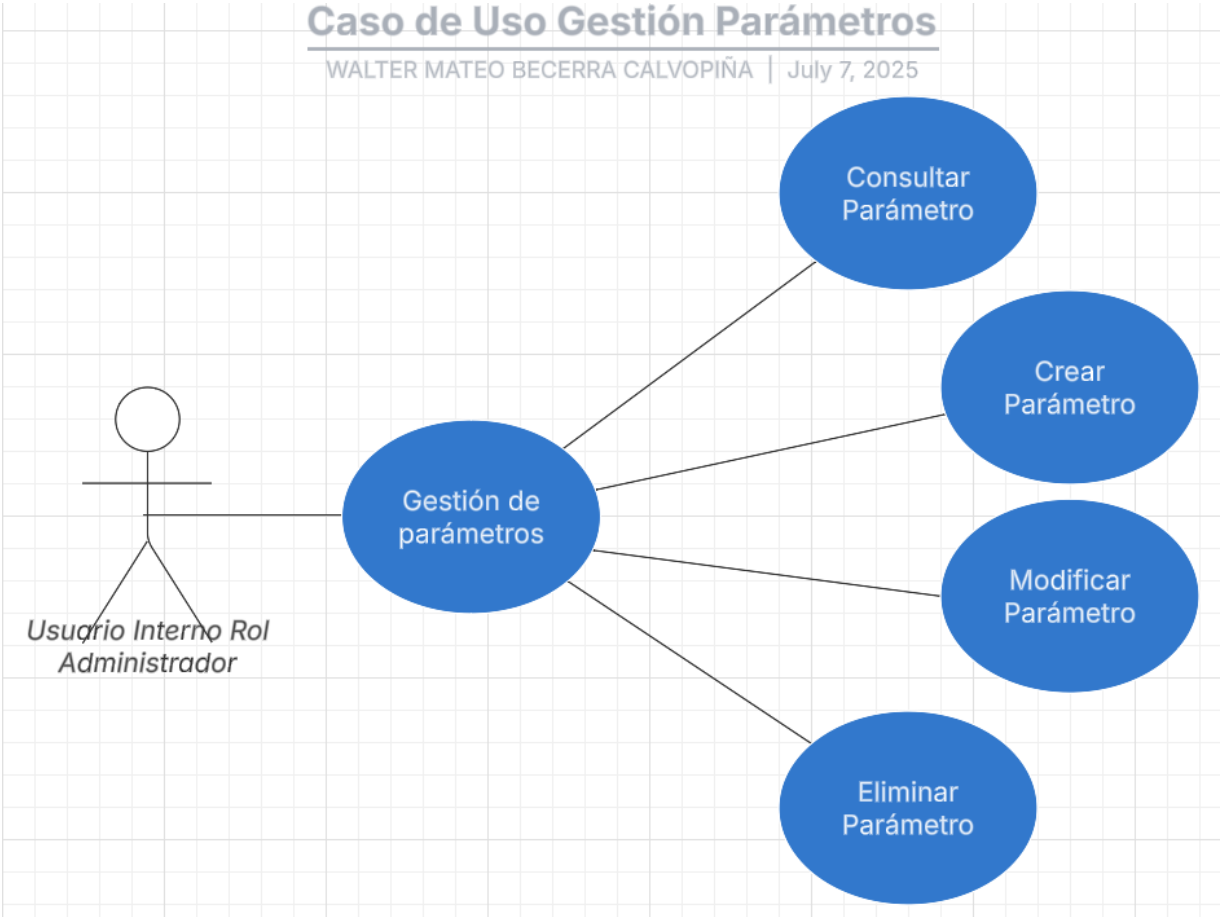
1. El usuario interno se dirige a la sección de Gestión de Parámetros.
2. Una vez dentro de la ventana de Gestión de Parámetros, el usuario interno procede a seleccionar en el Combo Box uno de los parámetros del sistema que se desea eliminar.
3. Una vez seleccionado el parámetro, se desplegarán los parámetros activos, se selecciona el ícono de eliminar “basurero color rojo”.
4. Se desplegará un modal donde se le indica al usuario con la siguiente pregunta” Esta seguro de que desea eliminar”
5. Se presiona el botón “Eliminar” y el sistema cambia de estado de activo a inactivo.

3.2.2.1.6 CONSULTAR PARÁMETRO

- **Actor:** Usuario Interno
- **Descripción:** En esta sección se le permite al usuario consultar un parámetro.
- **Flujo Principal:**

1. El usuario interno se dirige a la sección de Gestión de Parámetros.
2. Una vez dentro de la ventana de Gestión de Parámetros, el usuario interno procede a seleccionar en el Combo Box uno de los parámetros del sistema que se desea consultar.
3. El sistema reflejará la lista con los parámetros activos.
4. En esta vista se puede visualizar el nombre, el estado y las diferentes acciones que se pueden realizar.
5. El usuario debe presionar el icono de visualización “OJO”
6. Se desplegará un modal donde se puede visualizar los datos de ese parámetro.

Ilustración 3 Caso de Uso Gestión de Parámetros CU-001



Nota: Elaboración propia. Describe las acciones en la sección de Gestión de Parámetros.

Nombre del caso de uso: Gestión Parámetros	**ID Único:** CU-001
Área: --	
Actor(es): Usuario del Sistema, Administrador	
Interesados: Usuario del Sistema	
Nivel:	
Descripción: Permite a un usuario poder llevar una correcta gestión de los parámetros que se encuentran usando en todo el sistema, tanto en el área de primeras Consultas, trabajo social, actividades, gestión de usuarios, gestión de horarios.	
Evento desencadenador: El actor una vez se encuentre dentro del sistema, se dirige al menú de la parte izquierda, y pulsa la opción de "Gestión de Parámetros", desde esta vista se puede apreciar las distintas operaciones de CRUD.	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal):	**Información para los pasos**
1. El actor ingresa su correo y contraseña en el formulario de la página de login.	Correo electrónico, Contraseña
2. El sistema verifica el rol designado, para acceder a parámetros se acepta el rol administrador.	
3. El actor navega en el menú principal y hace clic en el botón "Gestión de Parámetros".	Ver la página de parámetros.
4. El actor navega en la vista de parámetros y puede realizar las distintas operaciones de CRUD por cada uno de los parámetros.	
Precondiciones: El usuario debe tener los permisos designados por el usuario administrador o ser el usuario administrador.	
Postcondiciones: El actor puede ver en el menú la gestión de parámetros y tener acceso a la vista de las operaciones CRUD.	
Suposiciones: El actor tiene los permisos necesarios para poder tener acceso a la vista de gestión de parámetros.	
Garantía de éxito: El actor puede ver la opción de "Gestión de Parámetros" y al hacer click se	

le presenta la vista de las operaciones de CRUD.
Garantía mínima: El sistema le proporciona la vista de gestión de parámetros pero no se presenta la vista.
Requerimientos cumplidos: Permitir que el actor pueda tener acceso a la gestión de parámetros.
Cuestiones pendientes: ¿Se implementarán nuevos parámetros, muy aparte de los existentes?
Prioridad: Alta
Riesgo: Alto

3.3 MODELADO DEL SISTEMA

En este apartado se presenta el modelado del sistema web "Balanza Web", en el cual se detallará mediante una representación gráfica que es clara y concisa de como se estructuran y relacionan los datos dentro del sistema. Con el fin de entender la interacción de las entidades involucradas, sus atributos claves y las relaciones existentes entre sí, fundamental para el correcto funcionamiento en el back-end.

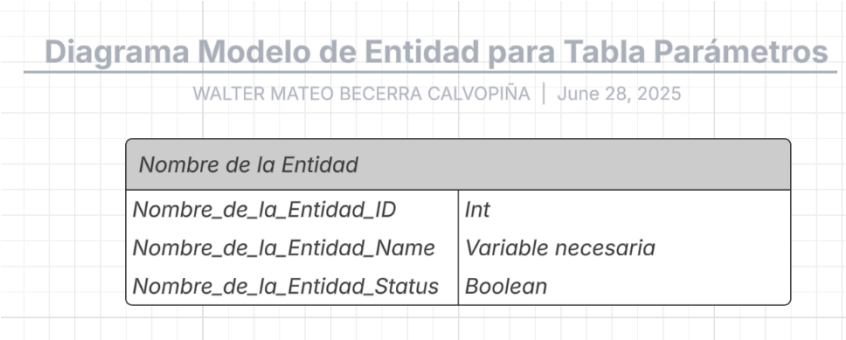
3.3.1 DIAGRAMA ENTIDAD RELACIÓN (ERD) ENFOQUE EN PARÁMETROS

Se pueden apreciar las 36 tablas de parámetros en la **Ilustración 3**, que apoyan al modelo entidad relación que se presentará más adelante en la gestión de todo el sistema, estas tablas se rigen al mismo esquema, se tienen 3 campos que mantienen el mismo formato que se detallará a continuación:

- El primero es el "Nombre_la_Tabla_ID" correspondiente a la clave primaria. Revisar la **Ilustración 4.**

- El segundo es el “Nombre_la_Tabla_Name” que corresponde al campo que será el nombre del parámetro Revisar la **Ilustración 4.**
- El tercero “Nombre_la_Tabla_Status” es el campo con el cual se empleará el borrado lógico del parámetro. Revisar la **Ilustración 4.**

Ilustración 4 Esquema Y Modelo de Tabla Parámetros



Nota: Elaboración propia. Esquema de la entidad tabla de parámetros.

Para un mejor entendimiento de la **Ilustración 5,** se va a seccionar las tablas por los módulos en donde se usan las mismas. Tenemos 3 módulos que engloban todo el sistema se detallará a continuación:

- Módulo de Primeras Consultas (Gestión de Casos)
  Es el primer filtro por donde pasan los usuarios, es la primera interacción donde se registra el caso y al usuario que requiera la consulta a los CJG. Las tablas que intervienen en Primeras consultas son las siguientes:

  - **Income_Levels:** Contiene los diferentes rangos de ingresos económicos que puede tener una persona.

- **Derived_Bies:** Registra los beneficios o apoyos que una persona ha recibido previamente. Definidos bajo los estándares de CJG.
- **Pensioners:** Identifica si una persona recibe una pensión del Estado o de otro organismo reconocido por el estado. Definidos bajo los estándares de CJG.
- **Health_Insurances:** Contiene los tipos de seguros de salud que posee el usuario.
- **Civil_Statuses:** Contiene todos los tipos de estados civiles. Definidos bajo los estándares de CJG.
- **Type_of_Attentions:** Describe las diferentes formas de atención que ofrece el consultorio, como presencial o virtual.
- **Own Assets:** Indica si la persona posee bienes propios. Definidos bajo los estándares de CJG.
- **Catastrophic_Ilnesses:** Enumera enfermedades graves que podrían afectar la situación legal o económica del usuario. Definidos bajo los estándares de CJG.
- **Complexities:** Clasifica los casos legales según su nivel de dificultad o especialización. Se basa en el expertis del usuario Interno para la elección del nivel de complejidad
- **Vulnerable_Situations:** Registra situaciones de vulnerabilidad social del usuario.
- **Type_of_Housings:** Detalla el tipo de vivienda que ocupa la persona. Definidos bajo los estándares de CJG.
- **Academic_Instructions:** Muestra el nivel de estudios alcanzado por el usuario. Definidos bajo los estándares de CJG.
- **Disabilities:** Identifica si la persona tiene alguna discapacidad registrada. Se mide porcentualmente una vez se escoja el tipo de discapacidad.
- **Ocupations:** Describe la profesión u ocupación del usuario.
- **Family_Incomes:** Registra el ingreso económico total del grupo familiar.
- **Ethnicities:** Indica el grupo étnico al que pertenece el usuario. Definidos bajo los estándares de CJG.

- **Subjects – Topics:** Lista los temas legales abordados en los casos, estas dos tablas van juntas ya qu están relacionadas directamente, esto debido a que el área tiene varias materias a tratar.
- **Countries - Provinces – Cities**: Ubicación geográfica del usuario o del caso legal. Estas tres van juntas ya que se relacionan entre sí.
- **Zone - Sector:** Segmenta las zonas, y cada una de las zonas contiene a los sectores de Quito.
- **Sexes:** Define el sexo o género del usuario. Definidos bajo los estándares de CJG.
- **Type_of_Attentions:** Aquí se define el tipo de caso puede ser solamente asesoría o derivarse a patrocinio. Contiene todos los tipos de estados civiles. Definidos bajo los estándares de CJG.

- Módulo Trabajo Social

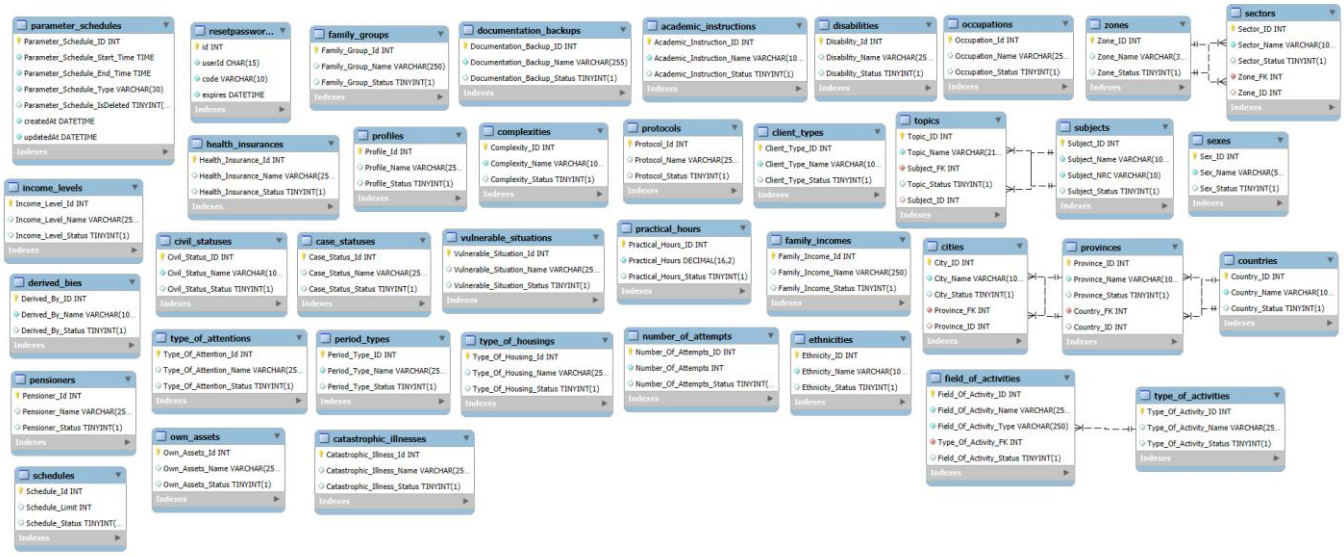
  - **Type_of_housings:** Detalla el tipo de vivienda que ocupa la persona. Definidos bajo los estándares de CJG.
  - **Civil_Statuses:** Contiene todos los tipos de estados civiles. Definidos bajo los estándares de CJG.

- Modulo de Control de Usuarios Internos (Gestión de Usuarios).

  - **Schedules:** Guarda los horarios de atención o disponibilidad del personal del consultorio.
  - **Period_Types:** Define los tipos de periodo usados en el sistema, como diario, semanal o mensual.
  - **Parameter_Schedules:** Relaciona parámetros específicos con ciertos horarios o franjas de atención.
  - **Resetpassword:** Contiene

- **Profiles:** Contiene los distintos tipos de roles que pueden tener los usuarios dentro del sistema.
- **Number_of_Attemps:** Contiene el número máximo de intentos que tiene cada usuario para recuperar su contraseña.

Ilustración 5 Modelo Entidad Relación - Parámetros

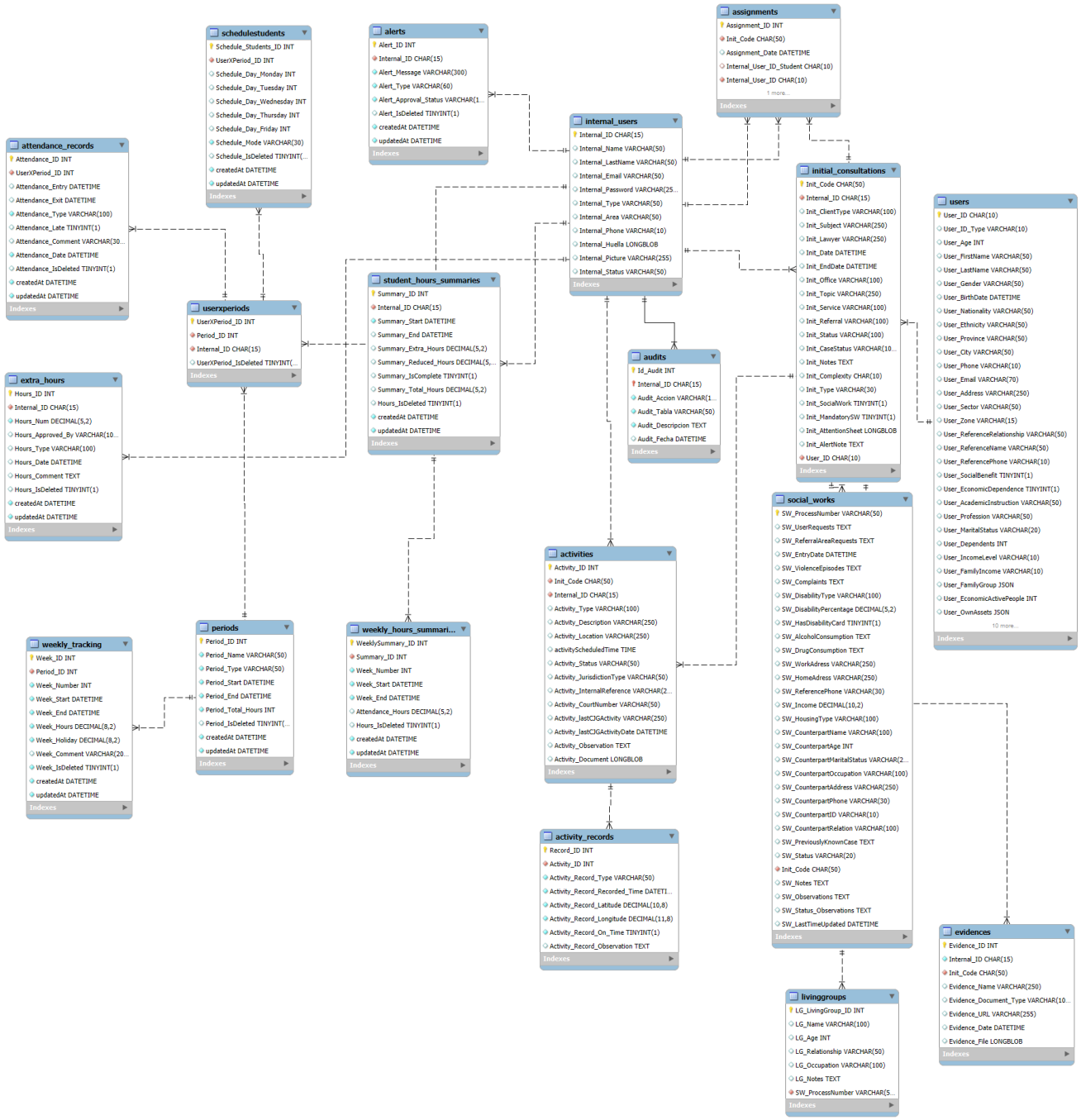


Nota: Elaboración propia. Ilustración de las 36 tablas de parámetros usadas en el sistema "Balanza Web"

3.3.2 MODELO ENTIDAD RELACIÓN (MER)

- El modelo entidad relación posee 19 tablas, como se muestra en la **Ilustración 6,** que controlan todo el sistema y cada uno de los módulos. A continuación, se dará la explicación del funcionamiento en conjunto por módulos y las interacciones de cada una de las tablas. Como se mencionó anterior mente se manejan 3 módulos. Módulo de Primeras Consultas (Gestión de Casos), módulo de Trabajo Social, módulo Control de Usuarios Internas.

Ilustración 6 Modelo Entidad Relación



Nota: Elaboración propia. Ilustración de las 19 tablas del MER usadas en el sistema “Balanza Web”

3.3.2.1 MÓDULO DE PRIMERAS CONSULTAS (GESTIÓN DE CASOS)

Este módulo está diseñado para gestionar las consultas iniciales que ingresan al Consultorio Jurídico Gratuito de la PUCE. Su funcionamiento integra varias entidades claves para el control y trazabilidad de cada caso. La tabla *"users", revisar la* **Ilustración 7,** almacena la información básica de los usuarios externos beneficiarios (clientes), mientras que *"internal_users", revisar la* **Ilustración 8,** contiene los datos del personal interno (pasante de CJG), como estudiantes y abogados que intervienen en los casos.

Las consultas como tal son registradas en la entidad *"initial_consultations", revisar la* **Ilustración 9,** donde se guarda información relevante del caso, estado y tipo de consulta. Las asignaciones de cada caso a los estudiantes son gestionadas en la tabla *"assignments", revisar la* **Ilustración 10,** que vincula a los usuarios con los abogados internos.

Además, las actividades asociadas al caso, tales como seguimientos, reuniones o trámites realizados, se documentan en la entidad *"activities", revisar la* **Ilustración 11,** todo este proceso se respalda con la tabla *"evidences", revisar la* **Ilustración 12,** que contiene archivos adjuntos cabe resaltar que únicamente se receptan archivos tipo .PDF que sea relevante al caso.

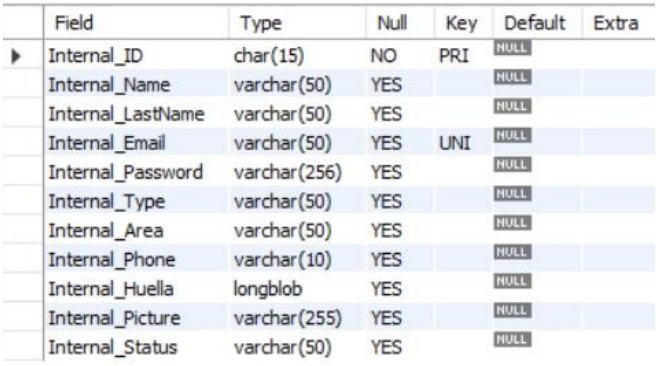
Para garantizar la integridad y trazabilidad del sistema, se incluye la entidad *"audits", revisar la* **Ilustración 13,** que permite llevar un control de cambios y acciones realizadas en el sistema (CRUD), indicando el usuario que modificó y la fecha correspondiente. En conjunto, este módulo permite gestionar de manera ordenada, segura y documentada cada una de las consultas jurídicas recibidas.

Ilustración 7 Entidad Users

	Field	Type	Null	Key	Default	Extra
▶	User_ID	char(10)	NO	PRI	NULL	
	User_ID_Type	varchar(10)	YES		NULL	
	User_Age	int	YES		NULL	
	User_FirstName	varchar(50)	YES		NULL	
	User_LastName	varchar(50)	YES		NULL	
	User_Gender	varchar(50)	YES		NULL	
	User_BirthDate	datetime	YES		NULL	
	User_Nationality	varchar(50)	YES		NULL	
	User_Ethnicity	varchar(50)	YES		NULL	
	User_Province	varchar(50)	YES		NULL	
	User_City	varchar(50)	YES		NULL	
	User_Phone	varchar(10)	YES		NULL	
	User_Email	varchar(70)	YES		NULL	
	User_Address	varchar(250)	YES		NULL	
	User_Sector	varchar(50)	YES		NULL	
	User_Zone	varchar(15)	YES		NULL	
	User_Referenc...	varchar(50)	YES		NULL	
	User_Referenc...	varchar(50)	YES		NULL	
	User_Referenc...	varchar(10)	YES		NULL	
	User_SocialBen...	tinyint(1)	YES		NULL	
	User_Economic...	tinyint(1)	YES		NULL	
	User_Academic...	varchar(50)	YES		NULL	
	User_Profession	varchar(50)	YES		NULL	
	User_MaritalSta...	varchar(20)	YES		NULL	
	User_Dependents	int	YES		NULL	

Nota: Elaboración propia. Entidad USERS con sus atributos.

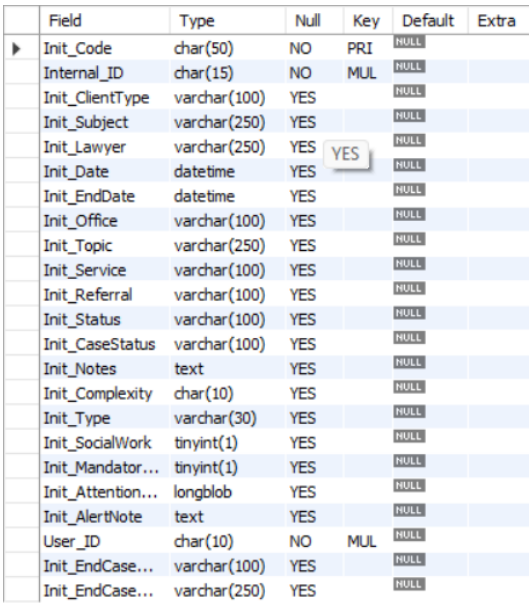
Ilustración 8 Entidad Internal_Users



	Field	Type	Null	Key	Default	Extra
▶	Internal_ID	char(15)	NO	PRI	NULL	
	Internal_Name	varchar(50)	YES		NULL	
	Internal_LastName	varchar(50)	YES		NULL	
	Internal_Email	varchar(50)	YES	UNI	NULL	
	Internal_Password	varchar(256)	YES		NULL	
	Internal_Type	varchar(50)	YES		NULL	
	Internal_Area	varchar(50)	YES		NULL	
	Internal_Phone	varchar(10)	YES		NULL	
	Internal_Huella	longblob	YES		NULL	
	Internal_Picture	varchar(255)	YES		NULL	
	Internal_Status	varchar(50)	YES		NULL	

Nota: Elaboración propia. Entidad Internal_Users con sus atributos.

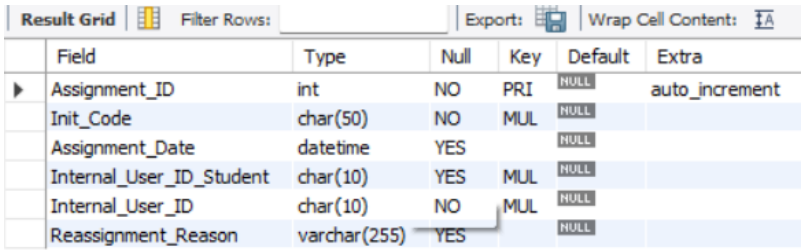
Ilustración 9 Initial_Consultations



	Field	Type	Null	Key	Default	Extra
▶	Init_Code	char(50)	NO	PRI	NULL	
	Internal_ID	char(15)	NO	MUL	NULL	
	Init_ClientType	varchar(100)	YES		NULL	
	Init_Subject	varchar(250)	YES		NULL	
	Init_Lawyer	varchar(250)	YES		NULL	
	Init_Date	datetime	YES		NULL	
	Init_EndDate	datetime	YES		NULL	
	Init_Office	varchar(100)	YES		NULL	
	Init_Topic	varchar(250)	YES		NULL	
	Init_Service	varchar(100)	YES		NULL	
	Init_Referral	varchar(100)	YES		NULL	
	Init_Status	varchar(100)	YES		NULL	
	Init_CaseStatus	varchar(100)	YES		NULL	
	Init_Notes	text	YES		NULL	
	Init_Complexity	char(10)	YES		NULL	
	Init_Type	varchar(30)	YES		NULL	
	Init_SocialWork	tinyint(1)	YES		NULL	
	Init_Mandator...	tinyint(1)	YES		NULL	
	Init_Attention...	longblob	YES		NULL	
	Init_AlertNote	text	YES		NULL	
	User_ID	char(10)	NO	MUL	NULL	
	Init_EndCase...	varchar(100)	YES		NULL	
	Init_EndCase...	varchar(250)	YES		NULL	

Nota: Elaboración propia. Entidad Initial_Consultations con sus atributos.

Ilustración 10 Entidad Assignments

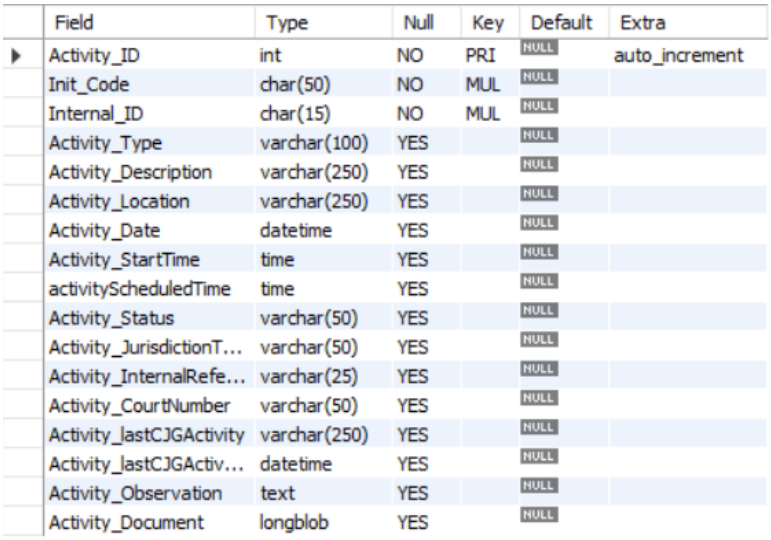


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Field	Type	Null	Key	Default	Extra
▶	Assignment_ID	int	NO	PRI	NULL	auto_increment
	Init_Code	char(50)	NO	MUL	NULL	
	Assignment_Date	datetime	YES		NULL	
	Internal_User_ID_Student	char(10)	YES	MUL	NULL	
	Internal_User_ID	char(10)	NO	MUL	NULL	
	Reassignment_Reason	varchar(255)	YES		NULL	

Nota: Elaboración propia. Entidad A*ssignments* con sus atributos.

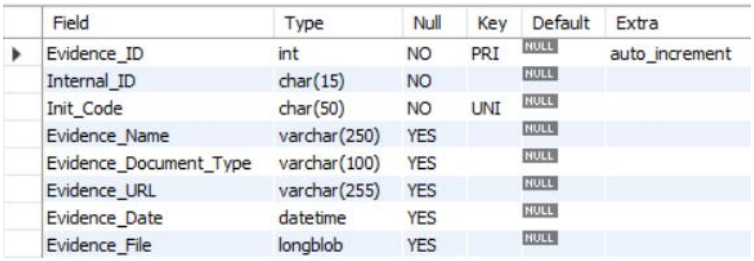
Ilustración 11 Entidad Activities



Field	Type	Null	Key	Default	Extra
Activity_ID	int	NO	PRI	NULL	auto_increment
Init_Code	char(50)	NO	MUL	NULL	
Internal_ID	char(15)	NO	MUL	NULL	
Activity_Type	varchar(100)	YES		NULL	
Activity_Description	varchar(250)	YES		NULL	
Activity_Location	varchar(250)	YES		NULL	
Activity_Date	datetime	YES		NULL	
Activity_StartTime	time	YES		NULL	
activityScheduledTime	time	YES		NULL	
Activity_Status	varchar(50)	YES		NULL	
Activity_JurisdictionT...	varchar(50)	YES		NULL	
Activity_InternalRefe...	varchar(25)	YES		NULL	
Activity_CourtNumber	varchar(50)	YES		NULL	
Activity_lastCJGActivity	varchar(250)	YES		NULL	
Activity_lastCJGActiv...	datetime	YES		NULL	
Activity_Observation	text	YES		NULL	
Activity_Document	longblob	YES		NULL	

Nota: Elaboración propia. Entidad *Activities* con sus atributos.

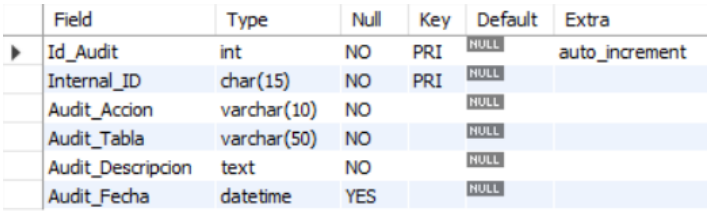
Ilustración 12 Entidad Evidences



Field	Type	Null	Key	Default	Extra
Evidence_ID	int	NO	PRI	NULL	auto_increment
Internal_ID	char(15)	NO		NULL	
Init_Code	char(50)	NO	UNI	NULL	
Evidence_Name	varchar(250)	YES		NULL	
Evidence_Document_Type	varchar(100)	YES		NULL	
Evidence_URL	varchar(255)	YES		NULL	
Evidence_Date	datetime	YES		NULL	
Evidence_File	longblob	YES		NULL	

Nota: Elaboración propia. Entidad *Evidences* con sus atributos.

Ilustración 13 Entidad Audits



	Field	Type	Null	Key	Default	Extra
▶	Id_Audit	int	NO	PRI	NULL	auto_increment
	Internal_ID	char(15)	NO	PRI	NULL	
	Audit_Accion	varchar(10)	NO		NULL	
	Audit_Tabla	varchar(50)	NO		NULL	
	Audit_Descripcion	text	NO		NULL	
	Audit_Fecha	datetime	YES		NULL	

Nota: Elaboración propia. Entidad Audits con sus atributos.

3.3.2.2 MÓDULO DE TRABAJO SOCIAL

El módulo de Trabajo Social se encarga de registrar, analizar y dar seguimiento a la situación socioeconómica de los usuarios que acuden al consultorio ya casos que ameriten el paso directo esto se reconoce con el expertis del usuario interno que se encuentra en primeras consultas. Está centrado en la entidad *"social_works",* revisar la **Ilustración 14**, la cual almacena la información recolectada por la trabajadora social, incluyendo condiciones de vivienda, entorno familiar, salud, y otros factores determinantes para la intervención.

El personal que lleva a cabo estas actividades," trabajadora social" se encuentra registrado en la tabla *"internal_users",* revisar la **Ilustración 8**, una vez revisado el caso se asigna . Adicionalmente, la entidad *"living_group",* revisar la **Ilustración 15***,* proporciona el detalle del grupo familiar con el que convive el usuario, permitiendo entender el contexto del hogar.

Todo análisis o intervención puede incluir documentos de respaldo, los cuales se almacenan en la tabla *"evidences",* revisar la **Ilustración 12***, l*a trazabilidad del módulo también está garantizada mediante la tabla *"audits"*, revisar la **Ilustración 13**, que

registra cada modificación o acción realizada en el sistema. Gracias a este conjunto de tablas, el módulo permite al área de Trabajo Social brindar un acompañamiento más integral a cada usuario, evaluando no solo aspectos legales, sino también humanos y sociales.

Ilustración 14 Entidad Social_Works

Field	Type	Null	Key	Default	Extra
SW_ProcessNumber	varchar(50)	NO	PRI	NULL	
SW_UserRequests	text	YES		NULL	
SW_ReferralAreaRequests	text	YES		NULL	
SW_EntryDate	datetime	YES		NULL	
SW_ViolenceEpisodes	text	YES		NULL	
SW_Complaints	text	YES		NULL	
SW_DisabilityType	varchar(100)	YES		NULL	
SW_DisabilityPercentage	decimal(5,2)	YES		NULL	
SW_HasDisabilityCard	tinyint(1)	YES		0	
SW_AlcoholConsumption	text	YES		NULL	
SW_DrugConsumption	text	YES		NULL	
SW_WorkAdress	varchar(250)	YES		NULL	
SW_HomeAdress	varchar(250)	YES		NULL	
SW_ReferencePhone	varchar(30)	YES		NULL	
SW_Income	decimal(10,2)	YES		NULL	
SW_HousingType	varchar(100)	YES		NULL	
SW_CounterpartName	varchar(100)	YES		NULL	
SW_CounterpartAge	int	YES		NULL	
SW_CounterpartMaritalSt...	varchar(20)	YES		NULL	
SW_CounterpartOccupation	varchar(100)	YES		NULL	
SW_CounterpartAddress	varchar(250)	YES		NULL	
SW_CounterpartPhone	varchar(30)	YES		NULL	
SW_CounterpartID	varchar(10)	YES		NULL	
SW_CounterpartRelation	varchar(100)	YES		NULL	
SW_PreviouslyKnownCase	text	YES		NULL	

Nota: Elaboración propia. Entidad Social_Works con sus atributos.

Ilustración 15 Entidad Living_Groups

Field	Type	Null	Key	Default	Extra
LG_LivingGroup_ID	int	NO	PRI	NULL	auto_increment
LG_Name	varchar(100)	YES		NULL	
LG_Age	int	YES		NULL	
LG_Relationship	varchar(50)	YES		NULL	
LG_Occupation	varchar(100)	YES		NULL	
LG_Notes	text	YES		NULL	
SW_ProcessNumber	varchar(50)	NO	MUL	NULL	

Nota: Elaboración propia. Entidad Living_groups con sus atributos.

3.3.2.3 MÓDULO DE GESTIÓN DE USUARIOS

Este módulo permite registrar y controlar las actividades realizadas por los estudiantes y usuarios internos del consultorio jurídico. El control se lleva a cabo por semanas y se estructura en base a distintos periodos definidos en la tabla *"periods"*, revisar la **Ilustración 16**, cada estudiante tiene un seguimiento específico almacenado en la entidad *"weekly_tracking"*, revisar las **Ilustración 17**, que registra los días y horas trabajadas semanalmente.

Además, la tabla *"userxperiod"*, revisar la **Ilustración 18**, asocia a cada estudiante con un periodo determinado, permitiendo vincular sus actividades con fechas exactas. En cuanto al desglose de horas, se lleva un control detallado en las tablas *"weekly_hours_summary"*, revisar la **Ilustración 19**, *"student_hours_summaries",* revisar la **Ilustración 20***,* y *"extra_hours"*, revisar la **Ilustración 21**, donde se contabilizan tanto las horas regulares como aquellas extraordinarias.

La tabla *"schedule_students"*, revisar la **Ilustración 22**, contiene los horarios asignados a cada usuario interno, permitiendo verificar la disponibilidad y cumplimiento de turnos. Finalmente, todo el módulo se encuentra auditado mediante la entidad "audits", revisar la **Ilustración 13**, asegurando que cada registro y modificación quede documentado. Este módulo es clave para monitorear el compromiso, cumplimiento y desempeño del personal interno dentro del consultorio.

Ilustración 16 Entidad periods

Field	Type	Null	Key	Default	Extra
Period_ID	int	NO	PRI	NULL	auto_increment
Period_Name	varchar(50)	NO		NULL	
Period_Type	varchar(50)	NO		NULL	
Period_Start	datetime	NO		NULL	
Period_End	datetime	NO		NULL	
Period_Total_Hours	int	NO		0	
Period_IsDeleted	tinyint(1)	YES		0	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Nota: Elaboración propia. Entidad periods y sus atributos.

Ilustración 17 Entidad Weekly_Tracking

Field	Type	Null	Key	Default	Extra
Week_ID	int	NO	PRI	NULL	auto_increment
Period_ID	int	NO	MUL	NULL	
Week_Number	int	NO		NULL	
Week_Start	datetime	NO		NULL	
Week_End	datetime	NO		NULL	
Week_Hours	decimal(8,2)	NO		0.00	
Week_Holiday	decimal(8,2)	NO		0.00	
Week_Comment	varchar(200)	YES		NULL	
Week_IsDeleted	tinyint(1)	NO		0	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Nota: Elaboración propia. Entidad Weekly_Tracking y sus atributos.

Ilustración 18 Entidad UserxPeriod

Field	Type	Null	Key	Default	Extra
UserXPeriod_ID	int	NO	PRI	NULL	auto_increment
Period_ID	int	NO	MUL	NULL	
Internal_ID	char(15)	NO	MUL	NULL	
UserXPeriod_IsDeleted	tinyint(1)	YES		0	

Nota: Elaboración propia. Entidad UserxPeriod

Ilustración 19 Entidad WeeklySummary

Field	Type	Null	Key	Default	Extra
WeeklySummary_ID	int	NO	PRI	NULL	auto_increment
Summary_ID	int	NO	MUL	NULL	
Week_Number	int	NO		1	
Week_Start	datetime	NO		NULL	
Week_End	datetime	NO		NULL	
Attendance_Hours	decimal(5,2)	YES		0.00	
Hours_IsDeleted	tinyint(1)	YES		0	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Nota: Elaboración propia. Entidad WeeklySummary

Ilustración 20 Entidad Student_Hours_Summaries

Field	Type	Null	Key	Default	Extra
Summary_ID	int	NO	PRI	NULL	auto_increment
Internal_ID	char(15)	NO	MUL	NULL	
Summary_Start	datetime	NO		NULL	
Summary_End	datetime	YES		NULL	
Summary_Extra_Hours	decimal(5,2)	YES		0.00	
Summary_Reduced_Hours	decimal(5,2)	YES		0.00	
Summary_IsComplete	tinyint(1)	YES		0	
Summary_Total_Hours	decimal(5,2)	YES		0.00	
Hours_IsDeleted	tinyint(1)	YES		0	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Nota: Elaboración propia. Entidad Student_Hours_Summaries y sus atributos.

Ilustración 21 Entidad Extra Hours

Field	Type	Null	Key	Default	Extra
Hours_ID	int	NO	PRI	NULL	auto_increment
Internal_ID	char(15)	NO	MUL	NULL	
Hours_Num	decimal(5,2)	NO		NULL	
Hours_Approved_By	varchar(100)	YES		NULL	
Hours_Type	varchar(100)	YES		NULL	
Hours_Date	datetime	YES		NULL	
Hours_Comment	text	YES		NULL	
Hours_IsDeleted	tinyint(1)	YES		0	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Nota: Elaboración propia. Entidad Extra hours y sus atributos.

Ilustración 22 Entidad Schedule_Students

Field	Type	Null	Key	Default	Extra
Schedule_Students_ID	int	NO	PRI	NULL	auto_increment
UserXPeriod_ID	int	NO	MUL	NULL	
Schedule_Day_Monday	int	YES		NULL	
Schedule_Day_Tuesday	int	YES		NULL	
Schedule_Day_Wednesday	int	YES		NULL	
Schedule_Day_Thursday	int	YES		NULL	
Schedule_Day_Friday	int	YES		NULL	
Schedule_Mode	varchar(30)	NO		NULL	
Schedule_IsDeleted	tinyint(1)	YES		0	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Nota: Elaboración propia. Entidad Schedule_Students y sus atributos.

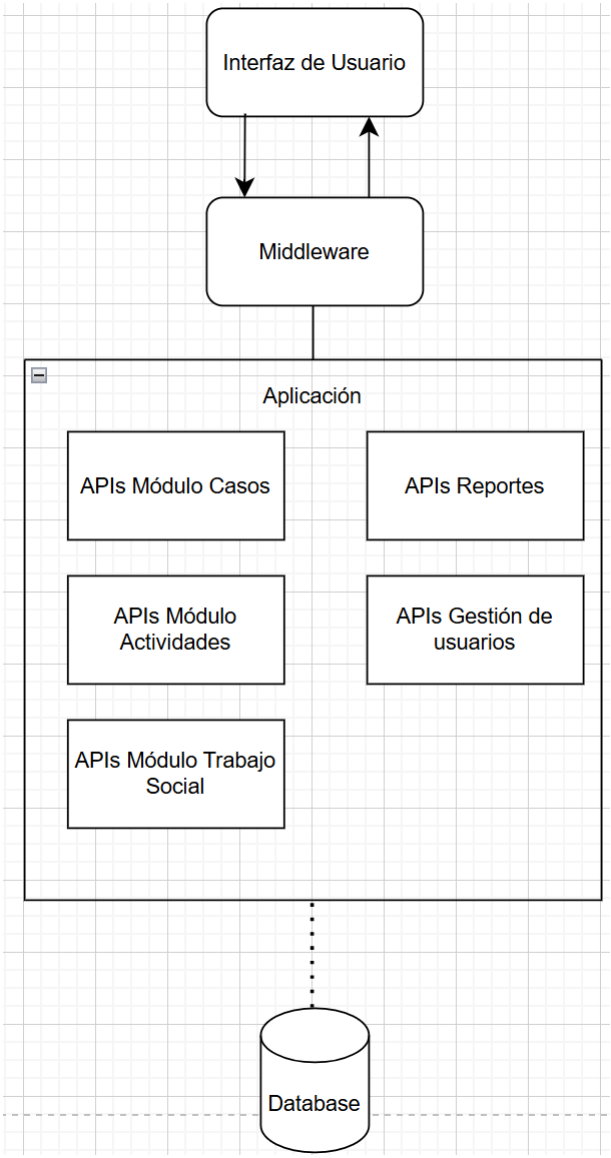
3.4 DISEÑO DE LA ARQUITECTURA DEL SISTEMA

La arquitectura del sistema, revisar la **Ilustración 23**, desarrollado para el CGJ. de la PUCE se construyó bajo el paradigma Cliente-Servidor, donde el usuario mediante un navegador web realiza peticiones al servidor, el cual procesa la lógica del negocio y responde con los datos solicitados. Esta arquitectura se complementa con una estructura basada en capas, lo que permite una clara separación de responsabilidades entre los distintos componentes del sistema, tales como la presentación, la lógica de negocio y el acceso a datos, tener la aplicación web de esta manera nos garantiza la mantenibilidad del sistema, permitiendo identificar y corregir errores de forma más ágil, sino que también mejora la escalabilidad, ya que cada capa puede evolucionar de manera independiente sin afectar al resto.

Para organizar el back-end se adoptó el patrón de diseño **Modelo-Vista-Controlador (MVC)**, hay que destacar que la VISTA en este caso se las maneja a través de las rutas para que puedan consumir todas las implementaciones del back-end, el cual no brinda un enfoque estructurado y reutilizable en la programación. Este patrón separa la lógica de acceso a datos (modelo), la lógica de control de flujo (controlador) y las rutas que sirven como interfaz de comunicación con el cliente. Gracias a esta implementación, el sistema es

más amigable para los desarrolladores y facilita el trabajo colaborativo y sienta una base sólida para futuras ampliaciones del sistema.

Ilustración 23 Diagrama Arquitectura del Sistema



Nota: Elaboración propia. Arquitectura de “Balanza Web”

3.4.1 DISEÑO DE LA BASE DE DATOS

La base de datos fue diseñada utilizando el motor de BD de **MySQL**, permitiendo estructurar la información de manera eficiente y segura. El modelo Entidad-Relación contempla las distintas entidades necesarias para los tres módulos del sistema:

- Primeras Consultas
- Trabajo Social
- Control de Usuarios Internos.

Se definieron relaciones claras entre las tablas para mantener la integridad de los datos y garantizar consistencia durante las operaciones del sistema.

Este diseño relacional permite realizar consultas eficientes, registrar evidencias documentales, asociar usuarios internos y externos con actividades específicas, y generar reportes personalizados. Todas las tablas del sistema están relacionadas con la entidad "audits", que registra de forma automática cada acción CRUD (crear, actualizar y eliminar) realizada por los usuarios. Esto garantiza un control exhaustivo de las operaciones, permitiendo rastrear quién realizó cada modificación y cuándo se llevó a cabo.
Justamente la tabla "audits" operá en las tablas de parámetros registrando de igual manera cada acción realizada

3.4.2 ESTRUCTURA DEL PROYECTO BACKEND (MVC)

La arquitectura del backend fue implementada siguiendo el patrón **Modelo -> Vista -> Controlador (MVC)**, con una división clara entre los distintos componentes del sistema, lo que permite mantener un desarrollo organizado, escalable y mantenible. Esta estructura está compuesta por las siguientes capas:

- **Schemas:** Esta capa representa la definición estructural de las tablas de la base de datos. En los archivos schema, se utilizan las funciones de Sequelize para declarar explícitamente cada campo, su tipo de dato, restricciones, relaciones foráneas y configuración adicional como claves primarias o timestamps. Esta capa actúa como una traducción directa del modelo relacional de la base de datos al entorno JavaScript. Revisar la **Ilustración 24.**

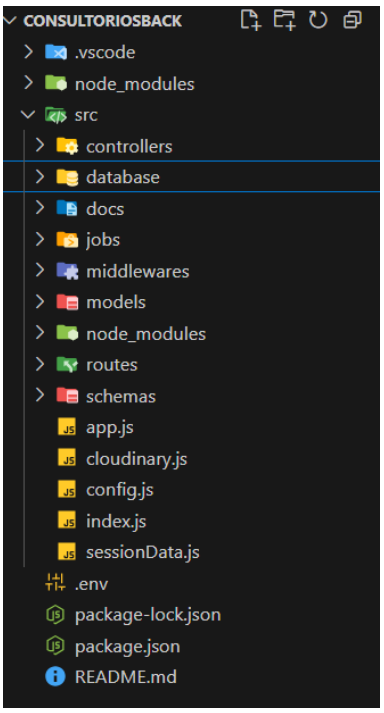
- **Models:** Contienen la lógica de negocio asociada a cada entidad. Aquí se implementan las funciones que permiten consultar, crear, actualizar, eliminar o buscar registros en la base de datos, utilizando los schemas como base. Los modelos también gestionan relaciones complejas, validaciones adicionales y operaciones específicas como generación de códigos, cálculos intermedios, manejo de archivos, y auditorías. Revisar la **Ilustración 24.**

- **Controllers:** Esta capa intermedia recibe las solicitudes desde el cliente (front-end) unicamente mediante peticiones HTTP, invoca los métodos del modelo correspondiente y envía una respuesta al usuario. Los controladores gestionan la lógica de flujo, control de errores y formateo de respuestas. Están organizados por módulos funcionales, como por ejemplo FirstConsultationsController para manejar las consultas jurídicas iniciales. Revisar la **Ilustración 24.**

- **Rutas (Routes):** Las rutas definen los endpoints del sistema y conectan cada solicitud entrante con su controlador respectivo. Están agrupadas según el módulo funcional que atienden, y muchas de ellas están protegidas por middleware de autenticación y validación de archivos. Por ejemplo, /initial-consultations permite registrar una nueva consulta y subir archivos adjuntos mediante la librería multer. Revisar la **Ilustración 24.**

- **Middlewares:** Se utilizan para tareas como la subida y validación de archivos, el manejo de errores, la protección de rutas sensibles mediante JWT y la transformación de datos. Estos componentes se ejecutan antes o después del controlador, ayudando a mantener el código modular y reutilizable. Revisar la **Ilustración 24**

Ilustración 24 Organización Back-End



Nota: Elaboración propia. Screenshot de Visual Studio Code.

4 CAPITULO IV: DESARROLLO DE LA APLICACIÓN

4.1 PREPARACIÓN DEL ENTORNO DE DESARROLLO

Para iniciar el desarrollo de la aplicación web, se configuró un entorno de trabajo moderno utilizando herramientas robustas y ampliamente adoptadas en el desarrollo backend. Se empleó Visual Studio Code como editor principal. El proyecto fue gestionado mediante códigos Git y almacenado en GitHub, permitiendo un control eficiente de versiones y trabajo colaborativo. Se definieron entornos de desarrollo, pruebas y producción.

Para inicializar el back-end de forma local es necesario tener en cuenta que se debe manejar las credenciales dentro del archivo de database. Justamente es para el entorno de desarrollo. Para el entorno de pruebas se procedió a mencionar a la DI – Departamento de Desarrollo, se encargaron de clonar los repositorios que se encuentran en GIT HUB, y se mantiene alojado en los servidores de la PUCE y se encuentra desplegado el ambiente de pruebas.

4.1.1 ESTÁNDARES DE CODIFICACIÓN

El desarrollo de la aplicación se realizó cumpliendo con los estándares establecidos por la Dirección de Informática – Departamento de Desarrollo (DI DESARROLLO) de la PUCE. Estas directrices fueron entregadas al inicio del proyecto y establecen lineamientos técnicos claros para garantizar la compatibilidad, escalabilidad y mantenimiento del sistema.

Entre los principales lineamientos establecidos por DI – DESARROLLO se destacan los siguientes:

- **Lenguaje y nomenclatura:** Toda la codificación debía realizarse en **inglés**, tanto en la estructura de la base de datos (nombres de tablas, campos, relaciones), como en el desarrollo del código fuente.

- **Motor de base de datos:** Se definió el uso obligatorio de MySQL**,** debido a que se mantuvieron reuniones en las cuales definimos el motor a usar**,** como sistema gestor de bases de datos relacional, por ser una solución de código abierto, estable, eficiente y alineada con los entornos tecnológicos de la PUCE.

- **Entorno de desarrollo:** Todo el backend debía ser desarrollado en Visual Studio Code, utilizando **Node.js** y JavaScript como lenguaje principal, respetando una arquitectura modular y orientada a servicios. Para el frontend, se estableció el uso del framework Vue.js, por su eficiencia en la creación de interfaces modernas y reactivas.

- **Control de versiones:** Se exigió el uso de **Git** para control de versiones y **GitHub** como plataforma para alojar el repositorio del proyecto. Esto permite mantener un historial detallado de los cambios realizados, facilitar la colaboración y garantizar la trazabilidad del desarrollo.

4.2 IMPLEMENTACIÓN DE LA FUNCIONALIDAD

La implementación del sistema se enfocó en desarrollar las funcionalidades necesarias para automatizar y optimizar los procesos del Consultorio Jurídico Gratuito. Se construyeron módulos independientes que permiten gestionar consultas iniciales, registrar evidencias documentales, generar hojas de atención, controlar el desempeño de los usuarios internos y dar seguimiento a los casos asignados. Cada uno de estos procesos fue estructurado mediante rutas, controladores y modelos específicos que interactúan entre sí de forma ordenada, garantizando así un flujo eficiente y coherente de información dentro del sistema.

Además, se integraron mecanismos de autenticación y autorización con tokens JWT para proteger los datos sensibles y restringir el acceso según el tipo de usuario. Se implementaron registros automáticos de auditoría para todas las operaciones críticas

(inserciones, actualizaciones y eliminaciones), asegurando la trazabilidad completa del sistema. La generación de reportes en formato PDF y Excel, la creación dinámica de códigos únicos para las consultas y la validación de reglas de negocio específicas también forman parte de la funcionalidad, todo ello diseñado para brindar soporte técnico eficiente al trabajo jurídico y social realizado por los estudiantes y personal del CJG.

4.2.1 BACKEND

El backend fue desarrollado con Node.js y el framework Express, bajo el patrón de arquitectura MVC. La definición de las estructuras de base de datos se gestionó en la carpeta schemas, mientras que la lógica de negocio reside en la capa de models. Los controllers manejan las solicitudes HTTP, y las routes conectan dichas solicitudes con las funciones correspondientes. Cada acción relevante en el sistema (crear, leer, actualizar, eliminar) es registrada automáticamente en la tabla audits para garantizar trazabilidad. Además, se utilizaron middlewares personalizados para autenticación JWT, manejo de archivos y validación de entradas.

4.3 PLAN DE PRUEBAS

Se elaboró un plan de pruebas orientado a validar la correcta funcionalidad del sistema, garantizando que cada módulo cumpla con su propósito y responda adecuadamente ante diferentes escenarios. Las pruebas se ejecutaron de forma controlada mediante herramientas como Postman y ThunderClient, simulando distintas solicitudes HTTP al backend.

4.3.1 TIPOS DE PRUEBAS

4.3.1.1 PRUEBAS DE UNIDAD

Las pruebas de unidad son un proceso fundamental que se centra en verificar el correcto funcionamiento de las partes más pequeñas del código, en este caso, las funcionalidades asociadas al módulo de gestión de parámetros. Este tipo de pruebas permite asegurar que cada función del sistema opere de forma aislada y se integre correctamente con el resto de componentes del backend. Como buena práctica de desarrollo, primero se construyó el sistema por bloques pequeños y se probaron de manera individual. Esto facilita la detección de errores, ya que permite identificar con precisión en qué parte específica del código ocurre una falla.

Durante el desarrollo, se realizaron pruebas unitarias sobre los endpoints de creación, modificación, consulta y eliminación lógica de parámetros, utilizando herramientas como Postman y ThunderClient. Inicialmente, las pruebas se llevaron a cabo únicamente sobre el backend, pero una vez integrado con el front-end, se validó también el comportamiento del sistema directamente desde la interfaz gráfica. Estas pruebas permitieron confirmar que las operaciones, sobre los parámetros se ejecutan correctamente, que los datos se almacenan con precisión y que cualquier cambio se registra automáticamente en el módulo de auditoría para mantener la trazabilidad del sistema.

4.3.2 CRITERIOS DE ACEPTACIÓN

Los criterios definidos para aceptar una funcionalidad incluyen:

- La información registrada debe almacenarse correctamente en la base de datos. Revisar la **Ilustración 25.**

- Toda operación debe ser registrada en la tabla audits. Revisar la **Ilustración 26.**

- Las relaciones entre tablas deben mantenerse intactas (integridad referencial).

Ilustración 25 Consulta a la BD

Internal_ID	Internal_Name	Internal_LastName	Internal_Email	Internal_Password	Internal_Type	Internal_Are
0000000000	Admin	PUCE	admin@puce.edu.ec	$2b$10$rt6UnVI7Ui5PIywoG9RLdumSO1I26d9MBOx8qvHYWbn7zk6EJQWo6	Administrador	Administrador
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Nota: Elaboración Propia. Consulta a la BD de un Registro en Internal_Users.

Ilustración 26 Registros en la tabla Audits

Id_Audit	Internal_ID	Audit_Acc	Audit_Tabla	Audit_Descripcion	Audit_Fecha
1	0000000000	INSERT	InternalUser	El usuario interno 0000000000 creó un nuevo registro de Usuario Interno con ID 1753...	2025-06-29 23:21:33
2	0000000000	INSERT	InternalUser	El usuario interno 0000000000 creó un nuevo registro de Usuario Interno con ID 1710...	2025-06-29 23:24:06
3	0000000000	INSERT	User	El usuario interno 0000000000 creó al usuario externo 1751617117	2025-06-29 23:36:48
4	0000000000	INSERT	Initial_Consultations	El usuario interno 0000000000 creó la consulta inicial AT-000001 para el usuario 1751...	2025-06-29 23:36:48
5	0000000000	INSERT	Evidences	El usuario interno 0000000000 subió la evidencia 1 para la consulta AT-000001	2025-06-29 23:36:48
NULL	NULL	NULL	NULL	NULL	NULL

Nota: Elaboración Propia. Consulta a la BD de los registros de Audits.

4.3.3 CASOS DE PRUEBA

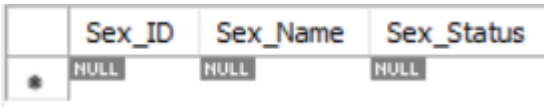
Los casos de prueba definidos en esta sección se enfocan en validar la correcta funcionalidad del backend del módulo de gestión de parámetros. Las pruebas se realizaron sobre los endpoints que permiten crear, actualizar, consultar y eliminar lógicamente un parámetro del sistema. Cada operación fue verificada mediante herramientas como Postman y ThunderClient, evaluando tanto la respuesta del servidor como el efecto real sobre la base de datos.

Para la prueba de la Gestión de parámetros debemos entender que al inicializar el back-end se crea automáticamente todas las tablas más un usuario administrador. Las tablas se encuentran completamente vacías, revisar **Ilustración 27**, por ende para que el sistema funcione hemos guardado en un bloc de notas el script de insert en la BD de todos los parámetros para la correcta gestión.

Una vez inicializado el sistema se procede a realizar el insert por SQL dentro de MYSQL. Y lo verificamos en el sistema mediante el front-end. **Revisar la Ilustración 28 y 29.**

Realizamos las pruebas respectivas de los métodos de Actualizar un registro, revisar la **Ilustración 30**, y Eliminar un registro, revisar la **Ilustración 31**.

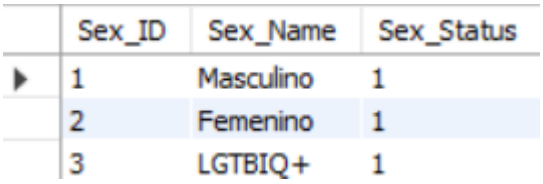
Ilustración 27 Captura Entidad Sexes



	Sex_ID	Sex_Name	Sex_Status
*	NULL	NULL	NULL

Nota: Elaboración Propia. Entidad de la Gestión de Parámetros Sexes Vacía

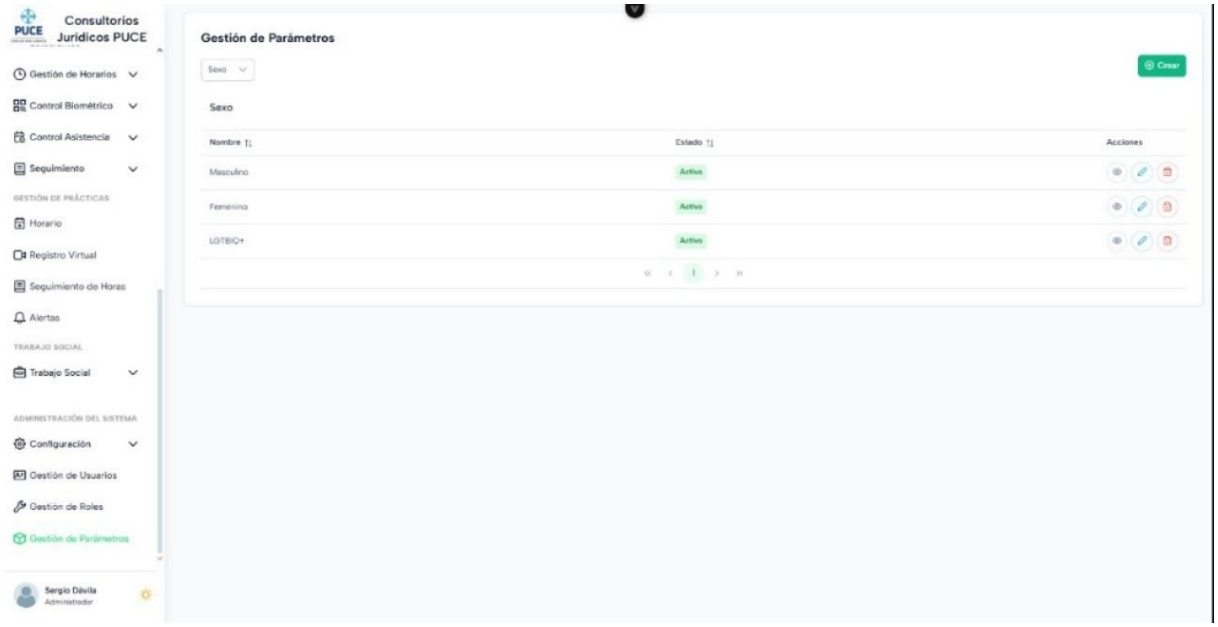
Ilustración 28 Captura Entidad Sexes



	Sex_ID	Sex_Name	Sex_Status
▶	1	Masculino	1
	2	Femenino	1
	3	LGTBIQ+	1

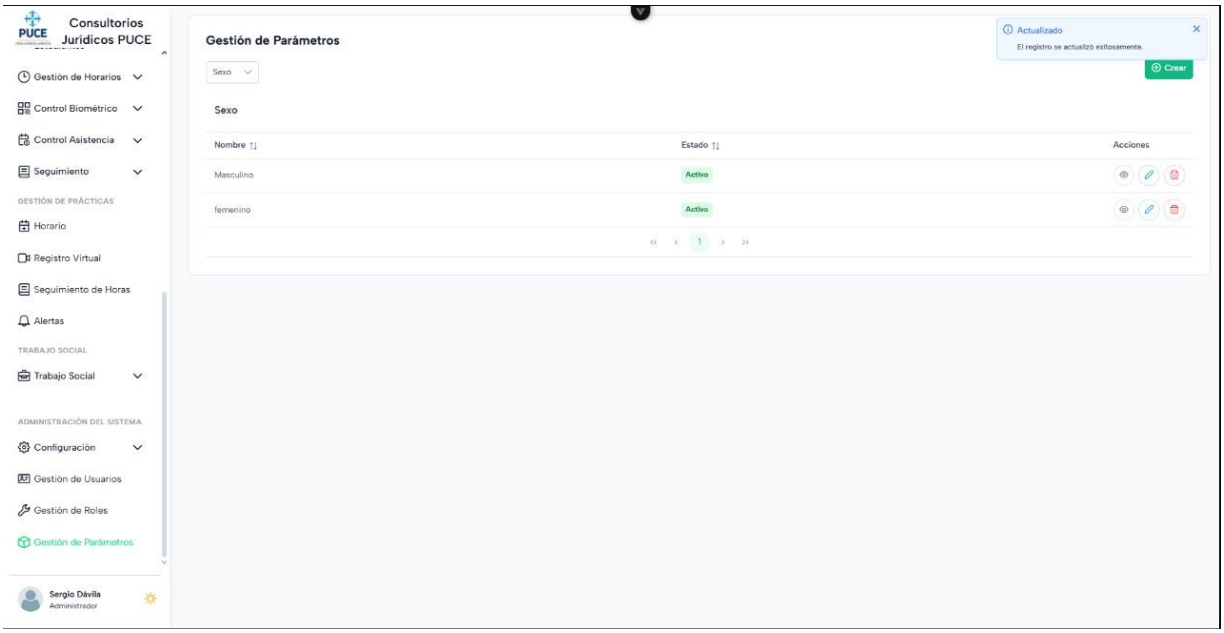
Nota: Elaboración Propia. Entidad de la Gestión de Parámetros Sexes con registros.

Ilustración 29 Captura Gestión Parámetros FRONT-END



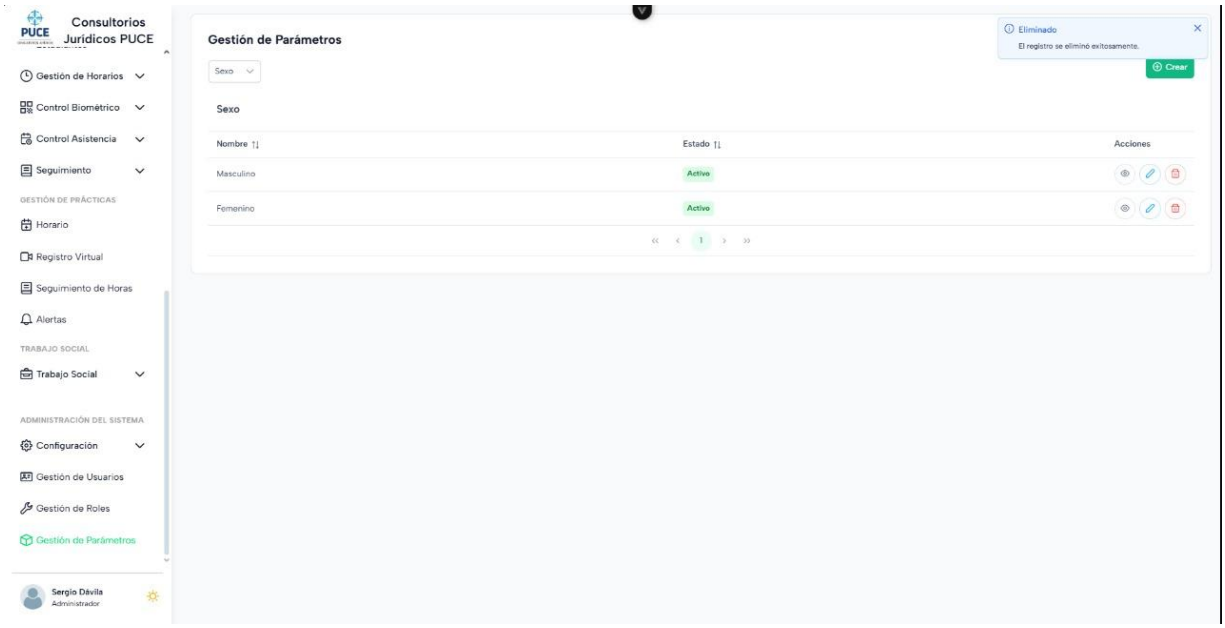
Nota: Elaboración Propia. Captura Front-End Gestión Parámetros después del Insert Masivo.

Ilustración 30 Captura Actualización de un Parámetro



Nota: Elaboración Propia. Apreciación de la actualización de un registro dentro de un Parámetro.

Ilustración 31 Captura Eliminación Lógica de un Parámetro



Nota: Elaboración Propia. Apreciación de la eliminación lógica de un parámetro.

4.4 DOCUMENTACIÓN DEL CÓDIGO

El módulo de gestión de parámetros representa uno de los componentes más críticos del backend del sistema, ya que permite mantener configuraciones dinámicas y reutilizables en múltiples formularios y procesos de la aplicación. Este módulo está conformado por un total de **36 tablas**, cada una encargada de almacenar los distintos catálogos del sistema, tales como sexo, estado civil, niveles de ingreso, tipos de atención, tipos de vivienda, entre otros.

Cada tabla de parámetros cuenta con su propio esquema declarado en la carpeta de "schema", donde se define su estructura mediante Sequelize. Un ejemplo de esto es la entidad Sex, en la cual se declara un campo entero como clave primaria "Sex_ID", un campo de texto obligatorio "Sex_Name" y un campo booleano "Sex_Status" que determina si el parámetro está activo o inactivo. Además, se desactivaron los timestamps por no ser requeridos para este tipo de entidades revisar la **ANEXO 1.**

A nivel de lógica, cada parámetro posee un modelo declarado en la carpeta “models” que encapsula su comportamiento. En estos modelos se implementan funciones específicas como create, update, delete y findAll, donde se integran validaciones previas, lógica de auditoría y gestión de errores. Revisar la **ANEXO 2, 3, 4 y 5,** para cada uno de los métodos correspondientes.

La capa de controladores declarado en la carpeta “controllers” recibe las solicitudes desde el cliente y canaliza la lógica hacia el modelo correspondiente. En este nivel se gestiona la creación individual o masiva de parámetros, la actualización y el retorno de mensajes de éxito o error, así como la respuesta HTTP adecuada según el caso. Las cabeceras también permiten identificar al usuario interno responsable de la acción, lo que se registra en la auditoría. Revisar la **ANEXO 6**.

Y por último las rutas declaradas en la carpeta “routes” permiten exponer las operaciones del back-end a través de endpoints protegidos. Cada tipo de parámetro cuenta con su propio archivo de rutas, lo que facilita la organización modular del sistema y su mantenibilidad a futuro. Estas rutas siguen convenciones RESTful y están preparadas para recibir peticiones mediante métodos GET, POST, PUT y DELETE. Revisar la **ANEXO 7.**

Este diseño modular aplicado al back-end de parámetros permitió replicar eficientemente la lógica en las 36 entidades, garantizando un comportamiento uniforme, auditable y fácilmente escalable.

La gestión de parámetros es el módulo que controla todo el sistema, ya que se parte de ahí para poder darle seguimiento a cada uno de los módulos. Un requerimiento funcional clave fue establecer una estructura flexible y centralizada para definir catálogos reutilizables, estos parámetros son fundamentales para el funcionamiento del sistema, ya que alimentan los formularios dinámicos, las validaciones de entrada, la lógica de negocio y las decisiones que se toman en módulos como Primeras Consultas, Trabajo Social y Control de Usuarios Internos. Sin este módulo, el sistema perdería coherencia y consistencia en la gestión de la información.

5 CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

El módulo de parámetros permitió centralizar la configuración dinámica del sistema, facilitando el control y la modificación de catálogos clave sin necesidad de alterar directamente la base de datos ni el código fuente.

La implementación del backend bajo el modelo MVC con Sequelize como ORM garantizó una estructura clara y mantenible para las operaciones CRUD de los parámetros, respetando las relaciones y validaciones requeridas.

Todas las operaciones realizadas sobre los parámetros fueron registradas automáticamente en la tabla de auditoría (audits), lo que asegura la trazabilidad total y fortalece la transparencia del sistema ante posibles revisiones o errores.

Se logró implementar una lógica de borrado lógico para los parámetros, lo que permite conservar el historial de registros desactivados sin eliminar datos de forma irreversible, protegiendo así la integridad referencial del sistema.

Las pruebas unitarias realizadas sobre el backend de parámetros demostraron que el sistema responde adecuadamente a distintos escenarios, validando los datos de entrada y generando respuestas claras y coherentes ante operaciones exitosas o fallidas.

5.2 RECOMENDACIONES

Mantener el módulo de parámetros correctamente documentado y sincronizado con los cambios funcionales del sistema, ya que cualquier modificación afecta directamente a otros módulos que dependen de estos valores.

Implementar un historial versionado para los parámetros, de modo que se pueda consultar cuándo y quién realizó cada modificación, mejorando así la trazabilidad y el control administrativo.

Desarrollar una funcionalidad que permita la carga y descarga masiva de parámetros mediante archivos en formato Excel o CSV, facilitando su gestión a nivel institucional.

Fortalecer las validaciones en las operaciones de creación y edición para evitar registros duplicados o mal estructurados que puedan generar errores de lógica en los formularios.

Visualizar de forma clara en el frontend el estado de cada parámetro (activo o inactivo), con el fin de evitar confusiones al momento de su selección en los formularios del sistema.

6 BIBLIOGRAFÍA

- Node.js. (2024). *About Node.js*. https://nodejs.org/en/about
- Sequelize. (2024). *Sequelize Documentation*. https://sequelize.org/
- JWT. (2024). *Introduction to JSON Web Tokens*. https://jwt.io/introduction/

- López, D., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web. http://138.59.13.30/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf
- MDN Web Docs. (2024). JavaScript Guide. Mozilla Developer Network.
- MySQL Documentation. (2024). MySQL Reference Manual. Oracle Corporation.
- Pontificia Universidad Católica del Ecuador. (2024). Manual de Buenas Prácticas en Consultorios Jurídicos. Quito, Ecuador.
- Aguilar, M. (2023). Desarrollo de Aplicaciones Web con Vue.js y Node.js. Editorial Técnica, Quito, Ecuador.
- ISO/IEC 27001:2013. (2024). Norma Internacional para la Gestión de Seguridad de la Información.

7 ANEXOS

ANEXO 1

Sex.js

src > schemas > parameter_tables > Sex.js > ...

```
1   import { DataTypes } from "sequelize";
2   import { sequelize } from "../../database/database.js";
3
4   export const Sex = sequelize.define('Sex', {
5       Sex_ID: {
6           type: DataTypes.INTEGER,
7           primaryKey: true,
8           autoIncrement: true
9       },
10      Sex_Name: {
11          type: DataTypes.STRING(50),
12          allowNull: false
13      },
14      Sex_Status: {
15          type: DataTypes.BOOLEAN,
16          defaultValue: true
17      }
18  }, { timestamps: false });
19
```

Nota: Elaboración propia. Ejemplo de Schema de la tabla de parámetros entidad SEX.

ANEXO 2

SexModel.js

src > models > parameter_models > SexModel.js > ...

```
4   export class SexModel {
26      static async create(data, internalId) {
27          try {
28              // Validar que el nombre del sexo no exista
29              const existingSex = await Sex.findOne({
30                  where: { Sex_Name: data.Sex_Name, Sex_Status: true }
31              });
32
33              if (existingSex) {
34                  throw new Error(`Sex with name "${data.Sex_Name}" already exists.`);
35              }
36
37
38              const newRecord = await Sex.create(data);
39
40                          await AuditModel.registerAudit(
41                              internalId,
42                              "INSERT",
43                              "Sex",
44                              `El usuario interno ${internalId} creó un nuevo registro de Sex con ID ${newRecord.Sex_ID}`
45                          );
46
47              return newRecord;
48
49          } catch (error) {
50              throw new Error(`Error creating sex: ${error.message}`);
51          }
52      }
```

Nota: Elaboración Propia. Captura del método créate del modelo de sexos.

ANEXO 3

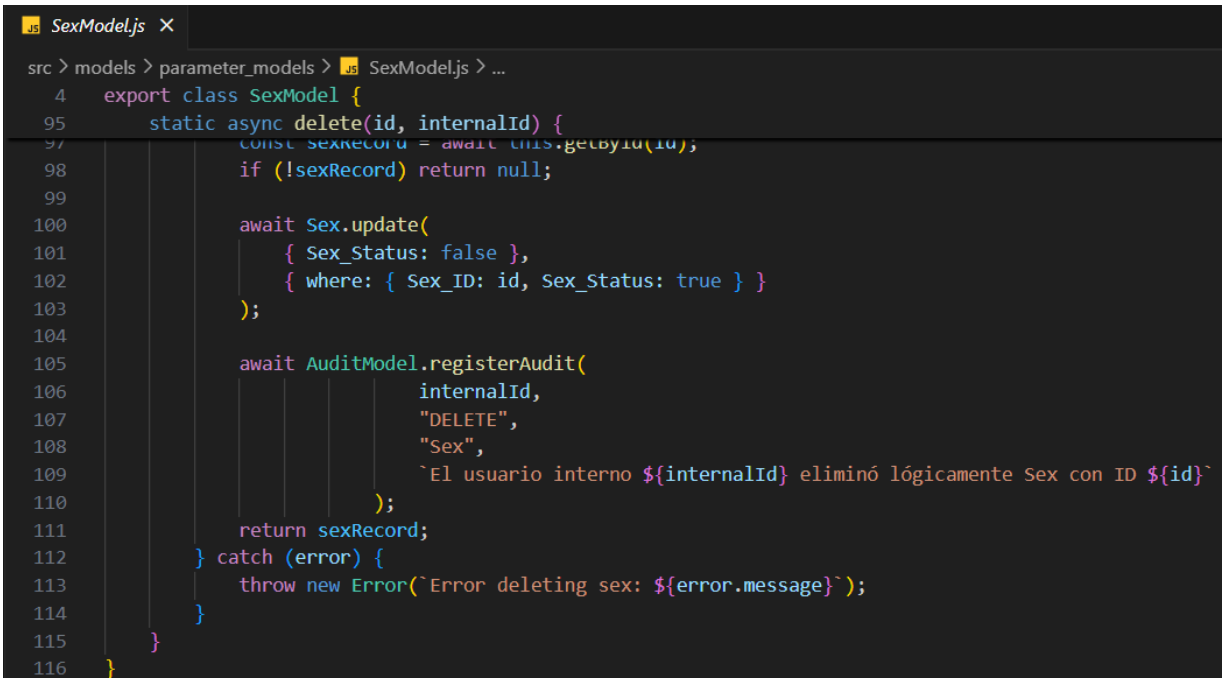
SexModel.js

src > models > parameter_models > SexModel.js > ...

```
4   export class SexModel {
70      static async update(id, data, internalId) {
71          try {
72              const sexRecord = await this.getById(id);
73              if (!sexRecord) return null;
74
75              const [rowsUpdated] = await Sex.update(data, {
76                  where: { Sex_ID: id, Sex_Status: true }
77              });
78
79              if (rowsUpdated === 0) return null;
80
81                          await AuditModel.registerAudit(
82                              internalId,
83                              "UPDATE",
84                              "Sex",
85                              `El usuario interno ${internalId} actualizó Sex con ID ${id}`
86                          );
87
88                          return await this.getById(id);
89
90          } catch (error) {
91              throw new Error(`Error updating sex: ${error.message}`);
92          }
93      }
```

Nota: Elaboración Propia. Captura del método update del modelo de Sexos.

ANEXO 4



Nota: Elaboración Propia. Captura del método delete del modelo de Sexos.

ANEXO 5

Nota: Elaboración Propia. Captura del método getAll del modelo de Sexos.

ANEXO 6

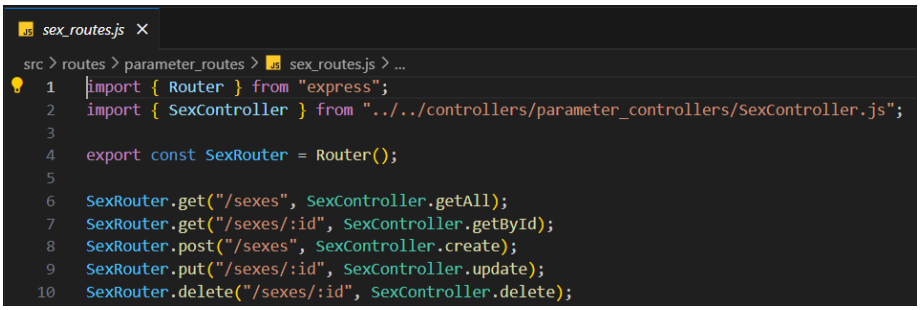
SexController.js

src > controllers > parameter_controllers > SexController.js > SexController > create

```
3   export class SexController {
25      static async create(req, res) {
26          try {
27              const internalId = req.headers["internal-id"]
28              if (Array.isArray(req.body)) {
29                  const createdSex = await SexModel.bulkCreate(req.body, internalId);
30                  return res.status(201).json(createdSex);
31              }
32              // Si es un objeto, usa create normal
33              const newSex = await SexModel.create(req.body, internalId);
34              res.status(201).json(newSex);
35          } catch (error) {
36              res.status(500).json({ error: error.message });
37          }
38      }
39      static async update(req, res) {
40          try {
41              const internalId = req.headers["internal-id"]
42              const { id } = req.params;
43              const updatedSex = await SexModel.update(id, req.body, internalId);
44              if (!updatedSex) return res.status(404).json({ message: "Sex not found or no changes made" });
45              res.status(200).json(updatedSex);
46          } catch (error) {
47              res.status(500).json({ error: error.message });
48          }
49      }
50      static async delete(req, res) {
51          try {
52              const internalId = req.headers["internal-id"]
53              const { id } = req.params;
54              const deletedSex = await SexModel.delete(id, internalId);
55              if (!deletedSex) return res.status(404).json({ message: "Sex not found" });
56              res.status(200).json(deletedSex);
57          } catch (error) {
58              res.status(500).json({ error: error.message });
59          }
60      }
```

Nota: Elaboración Propia. Captura de los tres métodos créate, update y delete con su respectivo manejo de errores.

ANEXO 7



Nota: Elaboración Propia. Captura de las rutas establecidas del parámetro Sex_routes.