PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

SISTEMAS DE INFORMACIÓN

TRABAJO DE TITULACIÓN

DESARROLLO DE UNA APLICACIÓN WEB PARA EL CONSULTORIO JURÍDICO GRATUITO DE LA PUCE - BACK END Y AUDITORIA

AUTOR

ESTEFANO ANDRÉS GALARZA ORTEGA

QUITO JULIO DE 2025

Dedicatoria

Dedico el presente trabajo de titulación a mis padres y hermano, personas que me dieron todo el apoyo, tanto en lo material como en lo emocional. Estuvieron ahi cuando más lo necesité y me dieron el valor y fuerza para llegar donde estoy ahorita. También dedico el presente trabajo de titulación a mi abuelita, que siempre estuvo a mi lado, dandome charlas que jamas olvidaré. Finalmente dedico el trabajo a Dayanara Martinez, una persona que a pesar de todos los conflictos que se presentaron, estuvo a mi lado, apoyandome con su cariño incondicional.

Agradecimiento

Agradezco a enormemete a mis padres por su apoyo y cariño incondicional hacia mi. Ellos me brindaron la grandiosa oportunidad de mejorar y ampliar mi formación académica. También agradezco a mis amigos, amigos que hice durante este camino y se convirtieron en personas cercanas con las que compartí sufrimiento y alegrías, desde salir de paseo hasta quedarse muchas madrugadas llorando por un punto y coma que faltaba en el código. Finalmente, agradezco a mí persona que después de tantos obstaculos y tantas batallas que peleaste, sigues de pie, no te rendiste en toda la trayectoría y sé que no te rendiras en el futuro.

Resumen

El presente trabajo de titulación tiene como objetivo desarrollar el backend y el módulo de auditoría de una aplicación web para el Consultorio Jurídico Gratuito de la Pontificia Universidad Católica del Ecuador (PUCE). Este desarrollo surge ante la necesidad de automatizar procesos internos que actualmente se gestionan de forma manual o desorganizada, afectando la eficiencia y trazabilidad del trabajo jurídico realizado por estudiantes y abogados. Para ello, se implementó una arquitectura basada en API RESTful, utilizando Node.js, Sequelize y una base de datos MySQL. Uno de los principales aportes del sistema es el módulo de auditoría, que registra automáticamente todas las acciones críticas realizadas por los usuarios internos (crear, actualizar, eliminar) sobre las diferentes entidades, asociando cada operación con el identificador interno (Internal_ID) del usuario autenticado. Además, se desarrollaron pruebas unitarias y funcionales para validar el correcto funcionamiento del módulo. El proyecto establece una base sólida para la continuidad de desarrollo en futuras etapas, mejorando así la calidad del servicio y la transparencia en el manejo de casos jurídicos.

Palabras clave: auditoría, backend, Node.js, consultorios jurídicos, API RESTful

Abstract

The purpose of this thesis is to develop the backend and audit module of a web application for the Free Legal Clinic of the Pontifical Catholic University of Ecuador (PUCE). This development arose from the need to automate internal processes that are currently managed manually or in a disorganized manner, affecting the efficiency and traceability of the legal work performed by students and lawyers. To this end, a RESTful API-based architecture was implemented using Node.js, Sequelize, and a MySQL database. One of the system's main contributions is the audit module, which automatically records all critical actions performed by internal users (create, update, delete) on different entities, associating each operation with the internal identifier (Internal_ID) of the authenticated user. In addition, unit and functional tests were developed to validate the module's correct operation. The project establishes a solid foundation for continued development in future stages, thereby improving the quality of service and transparency in the handling of legal cases.

Keywords: auditing, backend, Node.js, legal clinic, RESTful API.

Tabla de contenido

ÍNDICE DE GRÁFICOS

Anexos

CAPÍTULO I: INTRODUCCIÓN

1.1 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?

1.2 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.

1.3 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 (Sistema de auditoría).

1.4 Alcance

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

- Registro de casos: Creación de casos jurídicos con información detallada sobre cada proceso legal.
- Registro de evidencias de empleados: Gestión de documentos y evidencias en la base de datos.
- Auditoría: Implementación de mecanismos para rastrear y registrar acciones dentro del sistema, asegurando transparencia y trazabilidad en las operaciones.

El desarrollo seguirá la metodología de prototipado evolutivo, permitiendo iteraciones y mejoras progresivas basadas en pruebas y retroalimentación. Además, se adoptarán estándares y herramientas tecnológicos definidos por el Centro Informático de la PUCE para garantizar compatibilidad y calidad en la solución propuesta.

Finalmente, El presente proyecto no se completará en el lapso 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 a los consultorios jurídicos de la Pontificia Universidad Católica del Ecuador.

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

2.1 Aplicación Web y Servicios

Una aplicación web es un sistema o aplicación que se ejecuta en un navegador web. Comúnmente se lo utiliza para comunicarse con clientes para intercambiar información y brindar servicios de forma remota. La aplicación web se los puede ver en carros de compras, filtrado y búsqueda de productos, mensajería en línea, redes sociales, etc. Como diferencia principal a una aplicación local es que no es necesario instalar ningún software para el uso de la aplicación web, exceptuando el propio navegador.

Junto con una aplicación web, también se debe tomar en cuenta a la arquitectura de servicios. Esta arquitectura está compuesta por el proveedor y el consumidor, los cuales el proveedor es el que brinda servicios a través de un servidor y el consumidor es el cliente que los consume. El cliente se lo entiende como el FrontEnd y el servidor se lo entiende como el BackEnd. Para poder realizar esta comunicación, el BackEnd se expone a través de REST API, este mecanismo permite una comunicación continua y bidireccional, de tal forma que el cliente puede enviar y recibir datos del servidor y viceversa.

2.2 Metodología de desarrollo de software

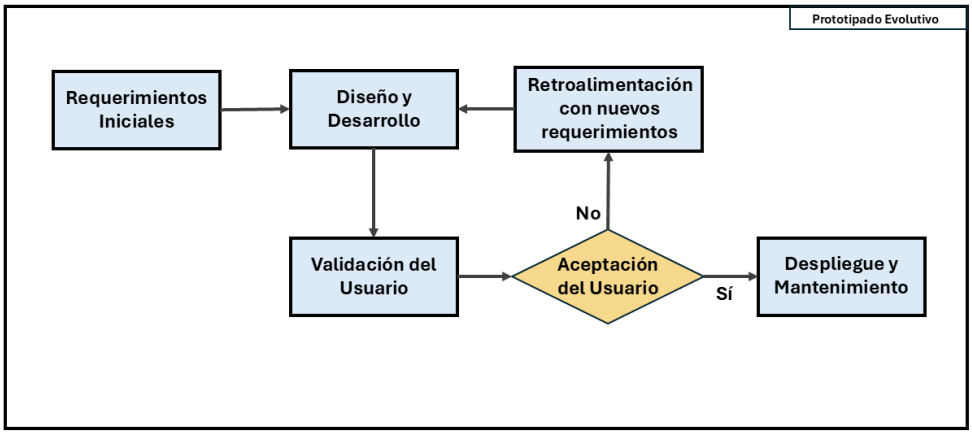
2.2.1 Prototipado Evolutivo

Se basa en la construcción de un prototipo de software que se construye rápidamente para que los usuarios puedan probarlo y aportar feedback. Así, se puede arreglar lo que está mal e incluir otros requerimientos que puedan surgir. Es un modelo iterativo que se basa en el método de prueba y error para comprender las especificidades del producto.

Es una metodología de desarrollo de software ágil en la cual se desarrolla un prototipo inicial de rápida construcción que gradualmente va evolucionando conforme se tienen comentarios de los stakeholders, comentarios que ayudan a la mejora y correcciones del proyecto. De este modo el desarrollo del sistema se vuelve flexible y escalable, incorporando parámetros e incluyendo al cliente en el proceso. Al inicio solo es un prototipo basado en requisitos previamente dichos en las que los clientes brindan feedback cada vez que se resuelva el anterior.

A continuación, se observa el diagrama de flujo de la metodología Prototipado evolutivo en la **Ilustración 1**.

ILUSTRACIÓN 1MODELO 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 motor de base de datos que nos permite gestionar las mismas, este es uno de los sistemas de código abierto más popular en el mundo. Las bases de datos son los espacios o repositorios en el que se almacena información, ya sea de empresas o de cualquier ente. Para esta gestión, se utiliza SQL, que significa "Structured Query Language", es un lenguaje que se encarga de consultar, actualizar, eliminar y gestionar bases de datos relacionales que utiliza un modelo cliente-servidor.

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 (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.

2.5 Entorno de Desarrollo Integrado (IDE)

2.5.1 Visual Studio Code

Visual Studio Code es una plataforma diseñada por Microsoft el cual se centra en edición de código, este software es de código abierto y de uso completamente gratuito. Existe un IDE con un nombre similar, Visual Studio, pero este se diferencia por su compatibilidad con tecnologías. Visual Studio está preparado para ser 100% compatible con aplicaciones propias de Microsoft, Visual Studio Code es capaz de adaptarse a cualquier lenguaje a partir de sus extensiones hechas por la comunidad.

2.6 Framework de desarrollo Backend

2.6.1 Node.Js

Node.js es un entorno de ejecución de un solo hilo, de código abierto y multiplataforma para crear aplicaciones de red y del lado del servidor 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 APIs. Se utiliza un formato JSON para transmitir información de manera confiable. Un JSON Web Token (JWT) está compuesto por tres partes fundamentales: el Header (encabezado), que especifica el tipo de token y el algoritmo de firma utilizado; el Payload (cuerpo o carga útil), que contiene la información que se desea transmitir, como la identidad del usuario y sus permisos; y la Signature (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 APIs.

2.7.2 Bcrypt

Bycript es una función de hash de contraseñas basada en el cifrado “Blowfish”. La derivación de claves es más lenta por lo que se la utiliza para que al atacante sea más lento el encontrar una contraseña. Bycript es un algoritmo de hashing de contraseñas diseñado para protegerlas frente a ataques maliciosos. A diferencia de otros métodos más antiguos como MD5, que son muy rápidos, bcrypt está pensado para ser de forma intencional, lento y seguro.

2.7.3 Multer

Multer es un middleware que se maneja mediante Node.Js para la carga y producción de archivos tipo foto, PDF, etc. Multer otorga a los desarrolladores la capacidad de especificar restricciones de tamaño de archivo, 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, este 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. 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. 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.

2.10 Buenas prácticas de desarrollo backend

2.10.1 Seguridad en APIs

La mayoría de las aplicaciones web modernas se basan en las API para funcionar, y las API introducen un riesgo adicional en una aplicación al permitir que partes externas accedan a ella. La seguridad de las API es el proceso de protegerlas de los ataques. Los ataques más habituales son: aprovechamientos de vulnerabilidades, ataques basados en la autenticación, errores de autorización y demasiadas solicitudes en corto tiempo. Como medida se puede poner un límite a la frecuencia con la que alguien puede repetir una acción dentro de un plazo determinado. Si un cliente de la API supera el número de solicitudes permitidas, la limitación de velocidad descartará o bloqueará sus solicitudes durante un periodo de tiempo.

2.10.2 Manejo de errores

Realizar una gestión correcta de las APIs ayuda a un fácil diagnóstico y resolución de problemas que se presenten. Por ejemplo, utilizar códigos de estado HTTP que se conozcan universalmente, error 404 significa que el recurso solicitado no existe. Otra manera es incluir mensajes de error detallados para que al momento en el que ocurra un error, se sepa de forma instantánea que es lo que sucede. Por último, la medida más recomendad es el realizar pruebas exhaustivas para asegurar que una API funcione de manera correcta.

2.11 Middleware

El middleware es un software que se centra en funcionar como puente entre las diferentes partes de un sistema completo, el middleware se encarga de comunicar tecnologías, herramientas y base de datos en un único sistema. Por ejemplo, el FrontEnd

de una aplicación recibe las interacciones del usuario, y este envía las solicitudes a un servidor externo del cual el usuario no tiene conocimiento, el middleware es el que se encarga de que esto se realice.

El middleware ofrece funciones que facilitan la comunicación entre diferentes aplicaciones y servicios mediante formatos de mensajería estándar como JSON, REST, XML, SOAP o servicios web. También suele incorporar mecanismos que permiten la interacción entre componentes desarrollados en diferentes lenguajes de programación.

CAPÍTULO III: ANALISIS Y DISEÑO DE LA APLICACIÓN

3.1 Análisis de requerimientos

Un análisis de requerimientos es la búsqueda de una necesidad y poder automatizarla mediante tecnología. En este caso, los Consultorios jurídicos gratuitos de la PUCE tenían un problema, un sistema antiguo incompleto, por lo que los empleados tendían que adoptar medidas improvisadas para ciertas tareas que podrían ser mejoradas. A través del análisis de requerimientos, se puede saber que procesos se pueden automatizar, y cómo hacerlo para que el trabajo de cada empleado sea más rápido y eficiente.

Para definir correctamente estos requerimientos, se realizaron reuniones periódicas con ambas partes, el equipo de desarrollo con los jefes de los Consultorios jurídicos gratuitos de la PUCE. Al acabar cada reunión se realizaron actas para constar todo, en donde se especificaron los siguientes aspectos:

- Fecha y hora de la reunión.
- Tema que se trató en la reunión.
- Participantes, aquí entraban todos los involucrados en la reunión.

- Objetivo de la reunión.
- Puntos que trataron en la reunión.
- Acuerdos logrados en la reunión.
- Acciones por seguir.
- Observaciones de la reunión.

Las actas de reuniones se basan en una misma plantilla como se observa en la **Ilustración 2**.

ILUSTRACIÓN 2 PLANTILLA ACTA DE REUNIÓN

ACTA DE REUNIÓN

No. #

PROYECTO DE DESARROLLO DE MÓDULOS DE JURISPRUDENCIA - PUCE

Fecha:	"Fecha de la reunion"	**Hora:**	"Hora de la reunion"
Tema:	"Tema de la reunión"		
1. Participantes "Participantes de la reunión"			
2. OBJETIVO "Objetivo de la reunión" **3. PUNTOS TRATADOS** "Puntos tratados de la reunión" **4. ACUERDOS LOGRADOS** "Acuerdos logrados por la reunión" **5. ACCONES POR SEGUIR** "Acciones por seguir para reuniones futuras" **6. OBSERVACIONES** "Observaciones de la reunión"			

Nota: Elaboración propia. Plantilla para contar las reuniones en actas.

Algunas de las reuniones predesarrollo fueron:

1. Reunión Introductoria del Proyecto – Participantes iniciales (4)

2. Separación de módulos y responsabilidades – 2 Profes y 7 participantes

3. Levantamiento de información– 7 participantes

4. Revisión de Diagramas de Casos de Uso – 2 Profes y 7 participantes

5. Reunión con la DI sobre tecnologías a usar (motor de bdd, lenguajes back y front) - 2 Profes, 7 participantes, DI

6. Reunión con DI para revisar avances y realizar corrección de errores - 2 Profes, 7 participantes, DI

7. Reunión con personal del consultorio jurídico para establecer requerimientos formales y presentación de mockups - 2 Profes, 7 participantes, Coordinadora de C.J, Dr. Enriquez

3.1.1 Requisitos Funcionales

- Almacenamiento de evidencias:

Actualmente, no existe ningún repositorio virtual o alguna forma para mantener centralizado los documentos proporcionados por los clientes y usuarios del sistema de "Balanza", todo se maneja por archivos en físico, por lo que esto ha creado sin fin de confusiones y una desorganización en relación con el manejo de archivos. Se sabe que también se manejan mediante Google drive, pero, aun así, esta no es una medida 100% efectiva. Como solución, se propuso la ingesta de evidencias en una base de datos, el nuevo sistema permitirá realizar consultas y cargar evidencia en un solo lugar.

- Implementación de sistema de Auditoria

Actualmente no existe ninguna forma de ver, ni evidencia alguna, sobre las acciones realizadas por los usuarios, tanto estudiantes como abogados sobre el sistema de "Balanza". Por lo que, si existiera algún error por parte de los usuarios, no quedaría constancia de este.

La implementación de esta funcionalidad ayudaría en este aspecto, se supiera todas las acciones (Crear, eliminar y actualizar) sobre cualquier entidad en la base de datos junto con la credencial de quien realizó dicha acción.

3.1.2 Requisitos no Funcionales

- Seguimiento de interacción de usuario con sistema

En un sistema robusto como es el nuevo sistema de Balanza web para consultorios jurídicos, se maneja un gran flujo de interacciones por parte de los usuarios internos. Por lo que todas estas interacciones deben quedar registradas de manera íntegra y certera. El acceso a auditoria no debe ser accesibles a cualquier usuario común. Para ello, se pueden ver los registros directamente en la base de datos.

- Fluidez

Balanza Web va a ser un sistema utilizado por toda el área de Consultorios Jurídicos gratuitos de la PUCE, por lo que tiene que ser de alta fluidez para que todos los usuarios, clientes y abogados, puedan gestionar los casos de manera eficiente, sin causar ningún estrés o mal sentimiento al usar el sistema.

3.2 Especificación de requerimientos

3.2.1 Diagramas casos de uso

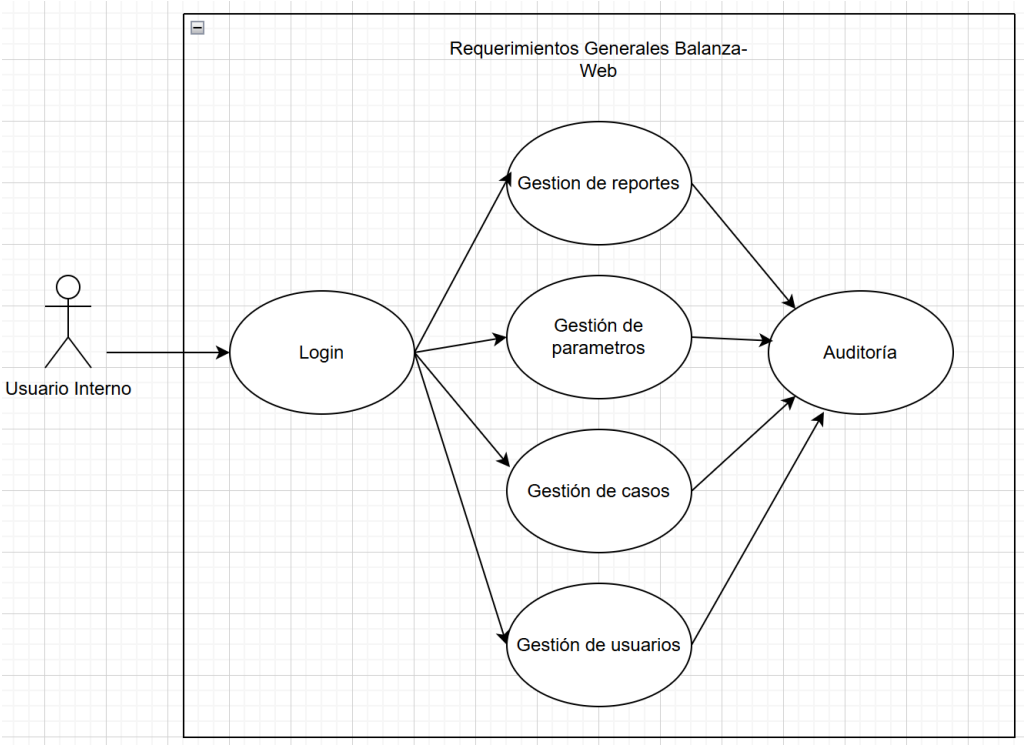
Un diagrama de caso de uso es una forma para plasmar en un diagrama la interacción que tiene los usuarios (Actores) con la funcionalidad del sistema. Se toma en cuenta las funcionalidades del sistema en sí, sin entrar mucho en detalles.

ID: CU01

Nombre: Requerimientos Generales Balanza-Web

Como se puede ver en la **Ilustración 3** se presenta la interacción total de un usuario interno con el sistema.

ILUSTRACIÓN 3 CASO DE USO REQUERIMIENTOS GENERALES: CU01



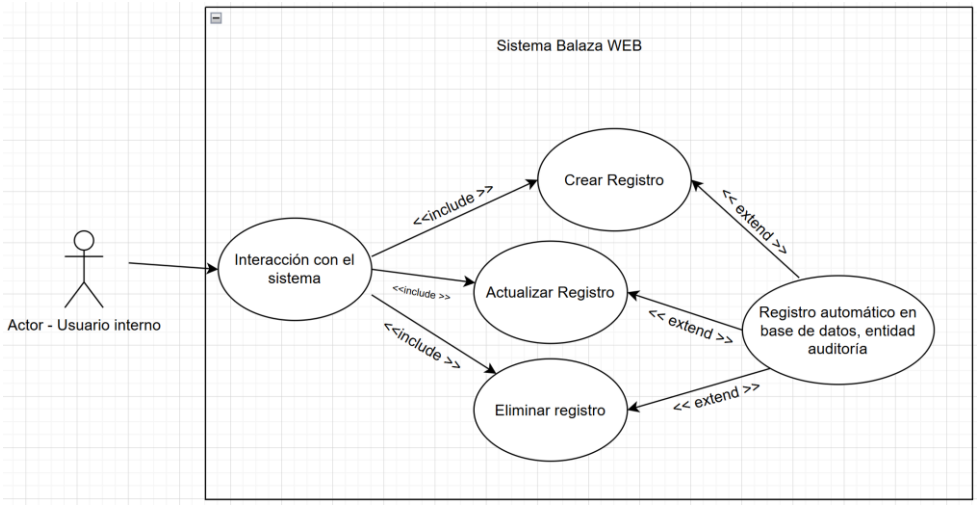
Nota: Elaboración propia. Diagrama caso de uso requerimientos generales

Nombre del caso de uso: Registro de auditoría	**ID Único:** CU-02
Área: -	
Actor(es): Usuario del Sistema, Administrador, Secretaría	
Interesados: Administradores del sistema	
Nivel: -	
Descripción: Este caso de uso es donde y el qué se va a almacenar en la entidad de Audits en la base de datos. El usuario tiene una interacción directa con el sistema, donde puede realizar acciones como crear, actualizar y eliminar registros de cualquier modulo. El usuario no tiene ninguna interacción con auditoria, el sistema es el que va ingresando en la entidad Audits de la base de datos todo lo que el usuario interno vaya realizando.	

Evento desencadenador: El actor navega a la página de Balanza Web (Login), introduce su correo electrónico y contraseña, y hace clic en el botón "Iniciar Sesión"	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal):	**Información para los pasos**
1. El usuario interno accede al sistema de forma correcta con sus credenciales dadas, este usuario tiene un rol con sus permisos específicos.	Correo electrónico, Contraseña
2. El usuario interno empieza con sus actividades respectivas dependiendo de su rol.	Información de actividades
3. Se procede a registrar todo movimiento de crear, actualizar y eliminar sobre los registros del sistema, todo se registra en la entidad Audits en la base de datos.	
Precondiciones: El usuario interno debe tener acceso al sistema y tener permisos necesarios para realizar las acciones de ingresar, actualizar y eliminar los registros de los módulos que sean necesarios.	
Postcondiciones: Se podrá revisar cualquier acción que haya realizado el usuario interno sobre los módulos del sistema de Balanza Web.	
Suposiciones: El actor tiene acceso a un navegador web compatible y conexión a la red de la PUCE.	
Garantía de éxito: El actor es redirigido a la página de inicio, visualiza el menú y procede con sus actividades correspondientes a su rol. Toda acción se registrará en la base de datos.	
Garantía mínima: El actor pudo iniciar sesión y procede a realizar las actividades, pero estas tal vez si o tal vez no se registran en la base de datos.	
Requerimientos cumplidos: Permitir que el actor pueda iniciar sesión eficientemente y se registran todos los movimientos en la base de datos.	
Cuestiones pendientes: ¿Se implementará una interfaz gráfica para que usuarios con permisos especiales puedan ver los datos de auditoría en el mismo sistema?	
Prioridad: Medio	

Riesgo: Medio

ILUSTRACIÓN 4 USUARIO INTERNO – AUDITORÍA: CU-02



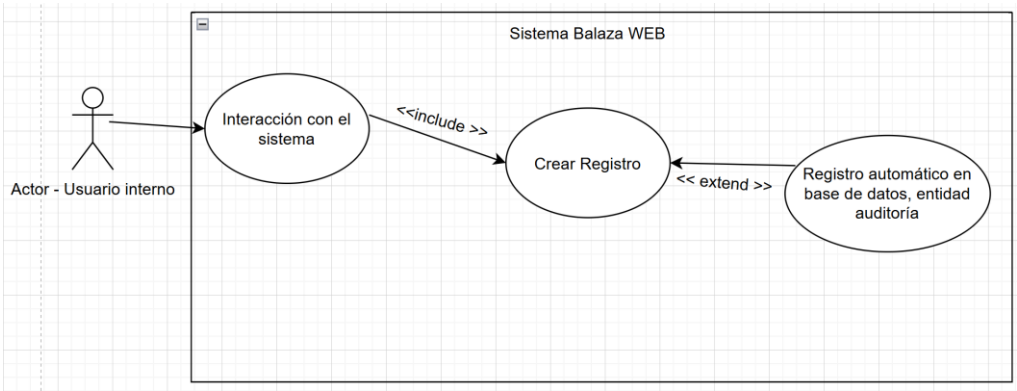
Nota: Elaboración propia. Diagrama caso de uso interacción del usuario interno con Auditoria.

ID: CU03

Nombre: Auditoria - Creación registros Balanza Web

En la **Ilustración 5** se observa la interacción de crear registro entre el usuario interno con auditoria.

ILUSTRACIÓN 5 CREACIÓN DE REGISTROS CON AUDITORIA: CU03



Nota: Elaboración propia. Diagrama caso de uso creación de elementos/registros del usuario interno sobre el sistema.

Actores Primarios:

- Usuario Interno: Tiene una interacción directa con el sistema, en cualquier módulo, el usuario interno puede crear registros ya sean casos, actividades, asignaciones de casos, gestión de usuarios

Descripción

Este caso de uso es donde y el qué se va a almacenar en la entidad de Audits en la base de datos con relación a la parte de creación. El usuario tiene una interacción donde puede crear diferentes elementos. Existen algunos elementos en el que se utilizan varias entidades de la base de datos de manera simultánea por lo que, en la creación de este tipo de elemento, auditoria capta el registro de estas tablas a la vez, por ejemplo, la creación de casos, este elemento en el sistema crea un usuario en la entidad Users, crea los datos del caso en la entidad Initial_consultations, y se crea una evidencia en la entidad de Evidencies. Claro que el usuario interno, dependiendo del rol, puede crear usuarios internos con sus respectivos roles, crear casos, asignar casos (En este caso se crea un registra en la entidad de Assigments), crear parámetros (Se crea un registro en la entidad respectiva de cada parámetro), y creación de actividades (Se crea un registro en la entidad Activities). El usuario no tiene ninguna interacción con auditoria directamente, el sistema es el que va ingresando en la entidad Audits de la base de datos todo lo que el usuario interno vaya creando.

Precondiciones:

- El usuario interno debe tener acceso al sistema y tener permisos necesarios para realizar las acciones de ingresar registros de los diferentes módulos que sean necesarios.

Flujo Principal:

- Ingreso al sistema

El usuario interno accede al sistema de forma correcta con sus credenciales dadas, este usuario tiene un rol con sus permisos específicos.

- Interacción con el sistema

El usuario interno empieza con sus actividades respectivas dependiendo de su rol, el usuario procede con la creación de casos, actividades, asignaciones, etc.

- Registro de interacción en Auditoria

Se procede a registrar todo movimiento de creación sobre los módulos de gestión mediante el sistema de auditoría. Se registra en la entidad Audits en la base de datos.

Flujo Alterno:

- Error de ingreso

El usuario interno no puede ingresar al sistema, por lo que no se puede registrar las acciones de este, el módulo de auditoria no recibe información sobre quién es el que quiere ingresar y, como no hay ingreso de usuario, tampoco puede realizar ninguna acción de crear registros del sistema.

Postcondiciones:

- Se podrá revisar cualquier creación que haya realizado el usuario interno sobre los módulos del sistema de Balanza Web.

Observaciones:

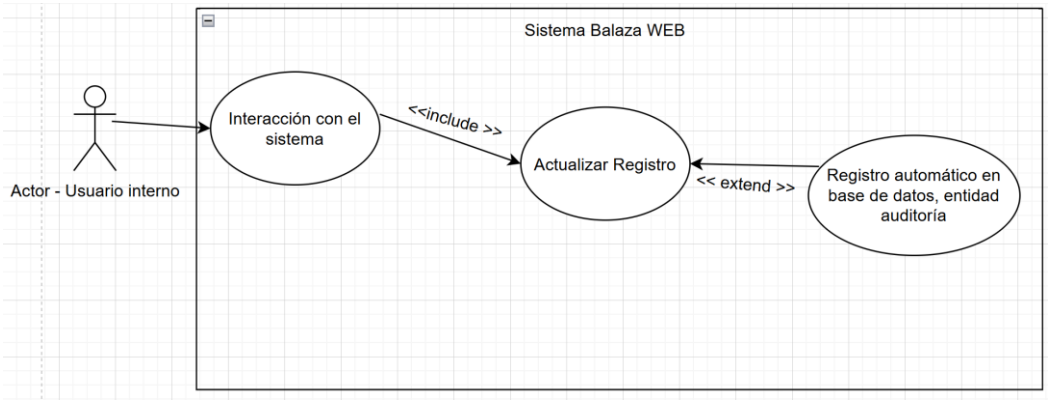
- El sistema debe registrar cualquier acción del usuario una vez haya ingreso con éxito al sistema, los registros solo se podrán observar directamente en la base de datos.

ID: CU04

Nombre: Auditoria - Actualizar registros Balanza Web

En la **Ilustración 6** se observa la interacción de actualizar un registro entre el usuario interno con auditoria.

ILUSTRACIÓN 6 ACTUALIZACIÓN DE REGISTROS CON AUDITORÍA



Nota: Elaboración propia. Diagrama caso de uso actualización de elementos/registros del usuario interno sobre el sistema.

Actores Primarios:

- Usuario Interno: Tiene una interacción directa con el sistema, en cualquier módulo, el usuario interno puede actualizar registros ya sean casos, actividades, asignaciones de casos, gestión de usuarios

Descripción

Este caso de uso es donde y el qué se va a almacenar en la entidad de Audits en la base de datos con relación a la parte de actualizar. El usuario tiene una interacción donde puede editar diferentes elementos ya creados. Existen algunos elementos en el que se utilizan varias entidades de la base de datos de manera simultánea por lo que, en la actualización de este tipo de elementos, auditoria capta el registro de estas tablas a la vez, por ejemplo la actualización de casos, en este elemento se puede editar un usuario en la entidad Users, edita los datos del caso en la entidad Initial_consultations, y puede actualizar una evidencia en la entidad de Evidencies. Claro que el usuario interno, dependiendo del rol, puede tener ciertas restricciones para actualizar usuarios internos con sus respectivos roles, crear casos, asignar casos (En este caso se edita un registra en la entidad de Assigments), crear parámetros (Se edita un registro en la entidad respectiva de cada parámetro), y creación de actividades (Se edita un registro en la entidad Activities). El usuario no tiene ninguna interacción con auditoria directamente, el sistema es el que va ingresando en la entidad Audits de la base de datos todo lo que el usuario interno vaya actualizando.

Precondiciones:

- El usuario interno debe tener acceso al sistema y tener permisos necesarios para realizar las acciones de editar registros de los diferentes módulos que sean necesarios.

Flujo Principal:

- Ingreso al sistema

El usuario interno accede al sistema de forma correcta con sus credenciales dadas, este usuario tiene un rol con sus permisos específicos.

- Interacción con el sistema

El usuario interno empieza con sus actividades respectivas dependiendo de su rol, el usuario puede o no editar casos, actividades, asignaciones, etc.

- Registro de interacción en Auditoria

Se procede a registrar todo movimiento de edición sobre los módulos de gestión mediante el sistema de auditoría. Se registra en la entidad Audits en la base de datos.

Flujo Alterno:

- Error de ingreso

El usuario interno no puede ingresar al sistema, por lo que no se puede registrar las acciones de este, el módulo de auditoria no recibe información sobre quién es el que quiere ingresar y, como no hay ingreso de usuario, tampoco puede realizar ninguna acción de crear registros del sistema.

Postcondiciones:

- Se podrá revisar cualquier actualización que haya realizado el usuario interno sobre los módulos del sistema de Balanza Web.

Observaciones:

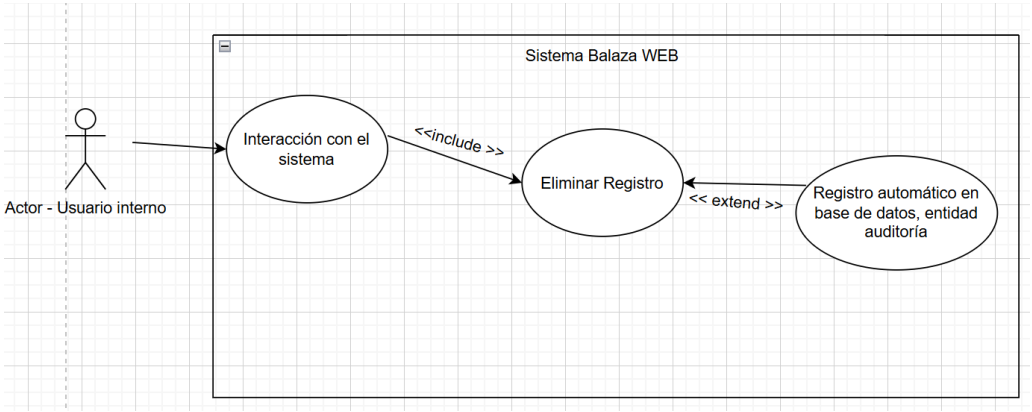
- El sistema debe registrar cualquier acción del usuario una vez haya ingreso con éxito al sistema, los registros solo se podrán observar directamente en la base de datos.

ID: CU05

Nombre: Auditoria - Eliminar registros Balanza Web

En la **Ilustración 7** se observa la interacción de eliminar un registro entre el usuario interno con auditoria.

ILUSTRACIÓN 7 ELIMINACIÓN DE REGISTROS CON AUDITORÍA



Nota: Elaboración propia. Diagrama caso de uso eliminación de elementos/registros del usuario interno sobre el sistema.

Actores Primarios:

- Usuario Interno: Tiene una interacción directa con el sistema, en cualquier módulo, el usuario interno puede eliminar registros ya sean casos, actividades, asignaciones de casos, gestión de usuarios, dependiendo de su rol.

Descripción

Este caso de uso es donde y el qué se va a almacenar en la entidad de Audits en la base de datos con relación a la parte de eliminación. El usuario tiene una interacción donde puede eliminar de manera logica diferentes elementos ya creados. Existen algunos elementos en el que se utilizan varias entidades de la base de datos de manera simultánea por lo que, en la eliminación de este tipo de elementos, auditoria capta el registro de estas tablas, por ejemplo, la eliminación de casos, en este elemento se puede eliminar un usuario en la entidad Users, se elimina los datos del caso en la entidad Initial_consultations, y puede eliminar una evidencia en la entidad de Evidencies. Claro que el usuario interno, dependiendo del rol, puede tener ciertas restricciones para eliminar usuarios internos con sus respectivos roles, eliminar casos, eliminación de parámetros (Se edita un registro en la entidad respectiva de cada parámetro y pone como estado inactivo), y eliminacion de actividades (Se edita un registro en la entidad Activities y se pone como estado en inactivo). El usuario no

tiene ninguna interacción con auditoria directamente, el sistema es el que va ingresando en la entidad Audits de la base de datos todo lo que el usuario interno vaya eliminando.

Precondiciones:

- El usuario interno debe tener acceso al sistema y tener permisos necesarios para realizar las acciones de eliminación de registros de los diferentes módulos que sean necesarios.

Flujo Principal:

- Ingreso al sistema

El usuario interno accede al sistema de forma correcta con sus credenciales dadas, este usuario tiene un rol con sus permisos específicos.

- Interacción con el sistema

El usuario interno empieza con sus actividades respectivas dependiendo de su rol, el usuario puede o no elimnar casos, actividades, asignaciones, etc.

- Registro de interacción en Auditoria

Se procede a registrar todo movimiento de eliminación sobre los módulos de gestión mediante el sistema de auditoría. Se registra en la entidad Audits en la base de datos.

Flujo Alterno:

- Error de ingreso

El usuario interno no puede ingresar al sistema, por lo que no se puede registrar las acciones de este, el módulo de auditoria no recibe información sobre quién es el que quiere ingresar y, como no hay ingreso de usuario, tampoco puede realizar ninguna acción de crear registros del sistema.

Postcondiciones:

- Se podrá revisar cualquier eliminación que haya realizado el usuario interno sobre los módulos del sistema de Balanza Web.

Observaciones:

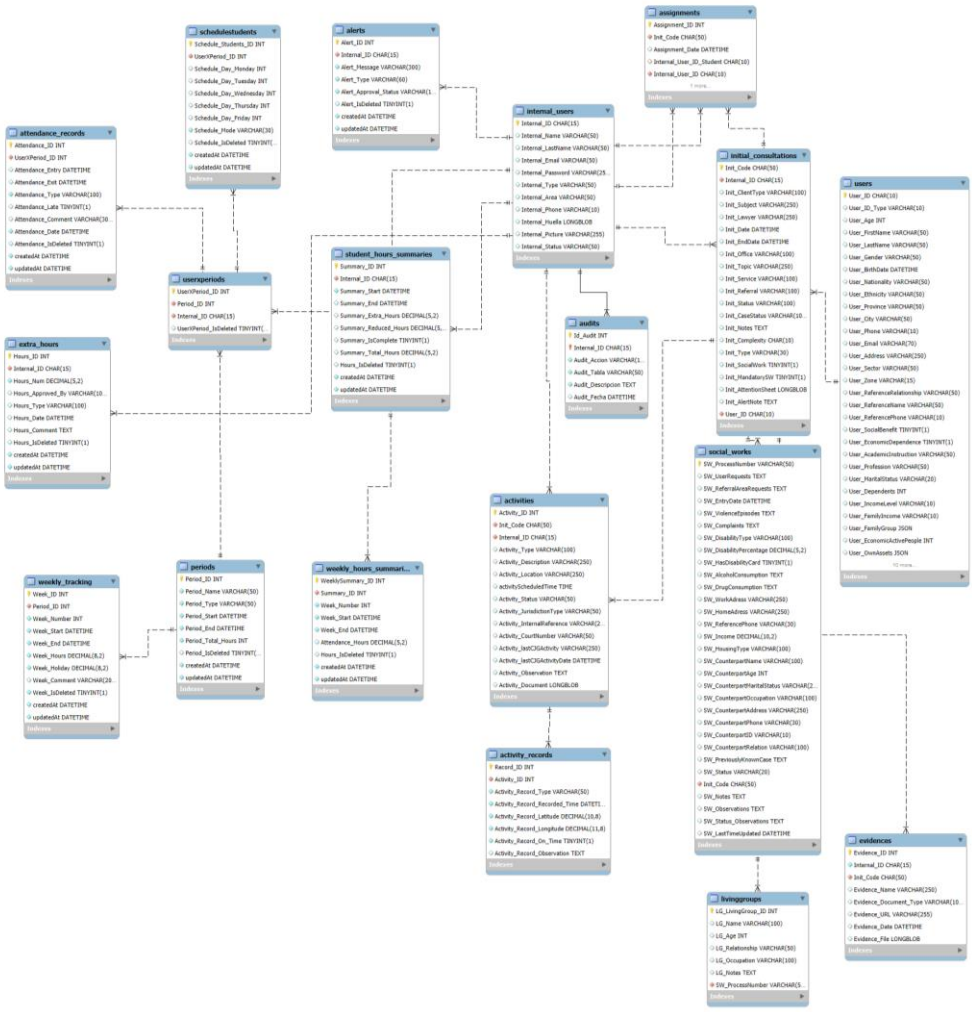
- El sistema debe registrar cualquier acción del usuario una vez haya ingreso con éxito al sistema, los registros solo se podrán observar directamente en la base de datos.

3.3 Modelado del sistema

3.3.1 Diagrama entidad relación (ERD)

En el siguiente diagrama **Ilustración 8**, se puede observar todas las entidades creadas para el sistema Balanza Web. La base de datos fue creada en base a requerimientos específicos del área de Consultorios jurídicos gratuitos de la PUCE.

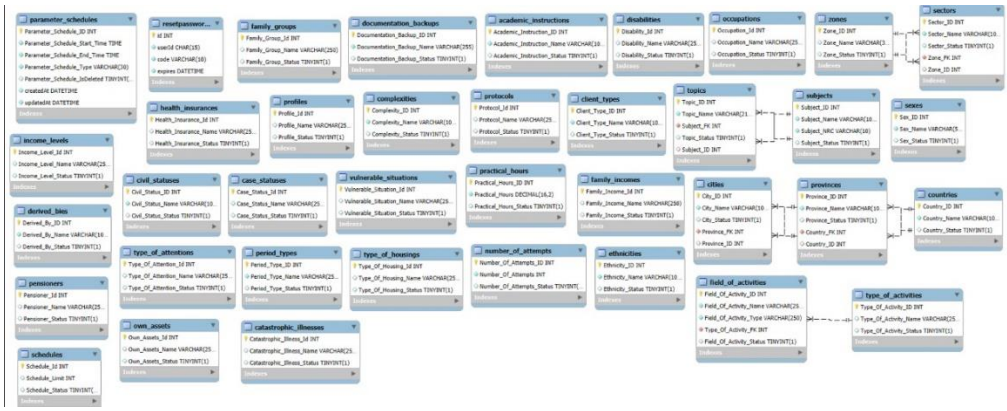
ILUSTRACIÓN 8 DIAGRAMA ENTIDAD RELACIÓN DE BASE DE DATOS



Nota: Elaboración propia. Diagrama entidad relación sobre la base de datos del sistema.

Como adición a las entidades principales, en consultorios jurídicos se requiere trabajar con ciertos parámetros. Por lo que en la **Ilustración 9**, se pueden ver las 36 entidades que son específicamente para ello, estas entidades entran en la categoría de "Tabla de parámetros". Existen dos excepciones, "resetpasswords" y "documentation_backups" que no entran en esta categoría.

ILUSTRACIÓN 9 DIAGRAMA ENTIDAD RELACIÓN ENTIDADES DE PARÁMETROS



Nota: Elaboración propia. Diagrama entidad relación sobre entidades de parámetros de la base de datos del sistema.

3.4 Diseño de la arquitectura del sistema

3.4.1 Diseño de la base de datos

La base de datos fue creada en el motor MySQL, siguiendo los principios de normalización.

Se tienen varios módulos principales:

- Módulo de auditoria utiliza la tabla “Audits” para el registro de auditorías en el sistema. La estructura de la tabla Audits se observa en la **Ilustración 10.**

ILUSTRACIÓN 10 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 en la base de datos

- Módulo de primeras consultas, se utilizan tres tablas de manera simultánea para la creación de casos, se utiliza: User (**Ilustración 11, Ilustración 12 e Ilustración 13**) para crear el cliente, Initial_Consultations (**Ilustración 14 e ilustración 15**) para crear el caso en sí, Evidences (**Ilustración 16**) para almacenar las evidencias subidas por el cliente.

ILUSTRACIÓN 11 ENTIDAD USERS PARTE 1

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	

Nota: Elaboración propia. Entidad Users Parte 1 en la base de datos

ILUSTRACIÓN 12 ENTIDAD USERS PARTE 2

Field	Type	Null	Key	Default	Extra
User_ReferencePh	varchar(10)	YES		NULL	
User_SocialBenefit	tinyint(1)	YES		NULL	
User_EconomicDep	tinyint(1)	YES		NULL	
User_AcademicInst	varchar(50)	YES		NULL	
User_Profession	varchar(50)	YES		NULL	
User_MaritalStatus	varchar(20)	YES		NULL	
User_Dependents	int	YES		NULL	
User_IncomeLevel	varchar(10)	YES		NULL	
User_FamilyIncome	varchar(10)	YES		NULL	
User_FamilyGroup	json	YES		NULL	
User_EconomicActi	int	YES		NULL	
User_OwnAssets	json	YES		NULL	
User_HousingType	varchar(50)	YES		NULL	
User_Pensioner	varchar(50)	YES		NULL	
User_HealthInsura	varchar(50)	YES		NULL	
User_VulnerableSit	varchar(50)	YES		NULL	

Nota: Elaboración propia. Entidad Users Parte 2 en la base de datos

ILUSTRACIÓN 13 ENTIDAD USERS PARTE 3

User_Disability	varchar(150)	YES	NULL	
User_DisabilityPerc	int	YES	NULL	
User_CatastrophicI	varchar(300)	YES	NULL	
User_HealthDocum	longblob	YES	NULL	
User_HealthDocum	varchar(150)	YES	NULL	
User_IsDeleted	tinyint(1)	YES	0	

Nota: Elaboración propia. Entidad Users Parte 3 en la base de datos

ILUSTRACIÓN 14 ENTIDAD INITIAL_CONSULTATIONS PARTE 1

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	

Nota: Elaboración propia. Entidad Initial_Consultations Parte 1 en la base de datos

ILUSTRACIÓN 15 ENTIDAD INITIAL_CONSULTATIONS PARTE 2

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 Parte 2 en la base de datos

ILUSTRACIÓN 16 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 en la base de datos

- Módulo de trabajo social, se utilizan tablas de Social_Work (**Ilustración 17 Ilustración 18**) junto con Living_Group (**Ilustración 19**).

ILUSTRACIÓN 17 ENTIDAD SOCIAL_WORKS PARTE 1

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(50)	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	

Nota: Elaboración propia. Entidad Social_Works primera parte en la base de datos.

ILUSTRACIÓN 18 ENTIDAD SOCIAL_WORKS PARTE 2

SW_CounterpartOccupation	varchar(100)	YES		NULL	
SW_CounterpartAddress	varchar(250)	YES		NULL	
SW_CounterpartPhone	varchar(50)	YES		NULL	
SW_CounterpartID	varchar(10)	YES		NULL	
SW_CounterpartRelation	varchar(100)	YES		NULL	
SW_PreviouslyKnownCase	text	YES		NULL	
SW_Status	varchar(50)	YES		Activo	
Init_Code	char(50)	NO	MUL	NULL	
SW_Notes	text	YES		NULL	
SW_Observations	text	YES		NULL	
SW_Status_Observations	text	YES		NULL	
SW_LastTimeUpdated	datetime	YES		NULL	

Nota: Elaboración propia. Entidad Social_Works segunda parte en la base de datos.

ILUSTRACIÓN 19 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 en la base de datos.

- Módulo de asignaciones, se utiliza la tabla Assignments (**Ilustración 20**).

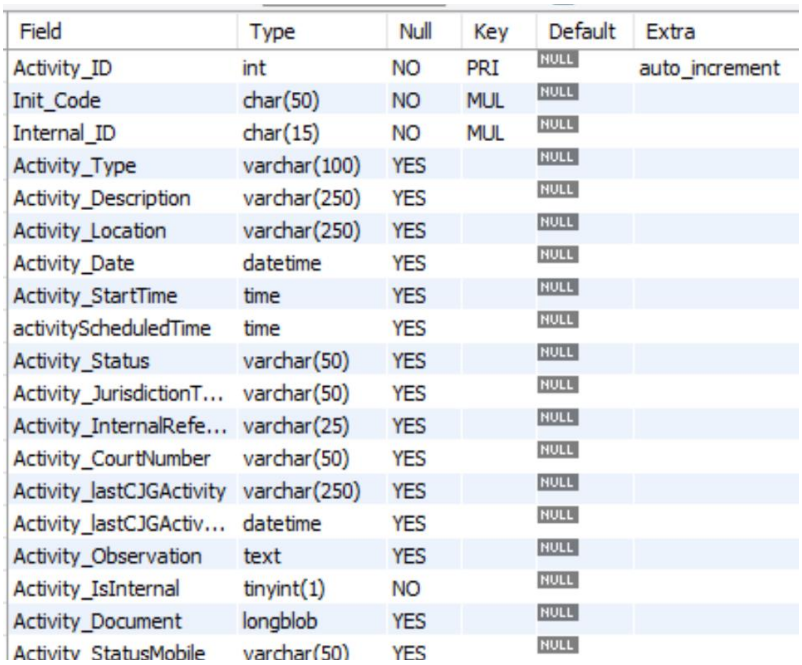
ILUSTRACIÓN 20 ENTIDAD ASSIGNMENTS

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 Assignments en la base de datos.

- Módulo de actividades, los abogados tienen el permiso para asignar actividades según el caso del estudiante que estén llevando en el momento. Para ellos se utilizan las entidades de Activities (**Ilustración 21**), Activity_records (**Ilustración 22**) y Evidences (**Ilustración 16**)

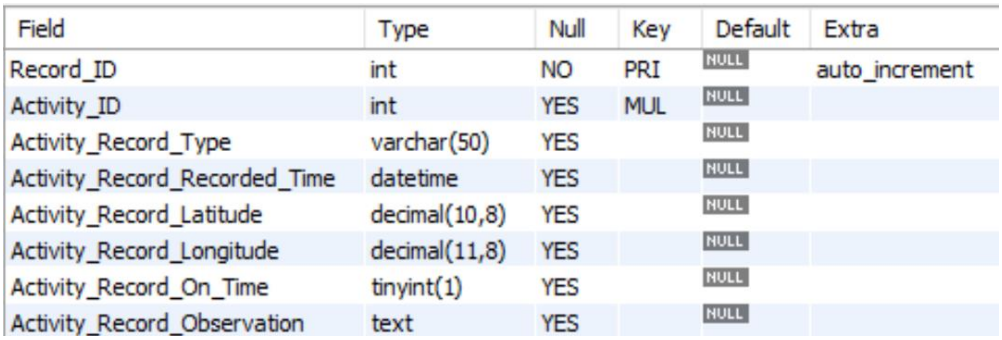
ILUSTRACIÓN 21 ENTIDAD ACTIVITIES



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_IsInternal	tinyint(1)	NO		NULL	
Activity_Document	longblob	YES		NULL	
Activity_StatusMobile	varchar(50)	YES		NULL	

Nota: Elaboración propia. Entidad Activities en la base de datos.

ILUSTRACIÓN 22 ENTIDAD ACTIVITY_RECORDS



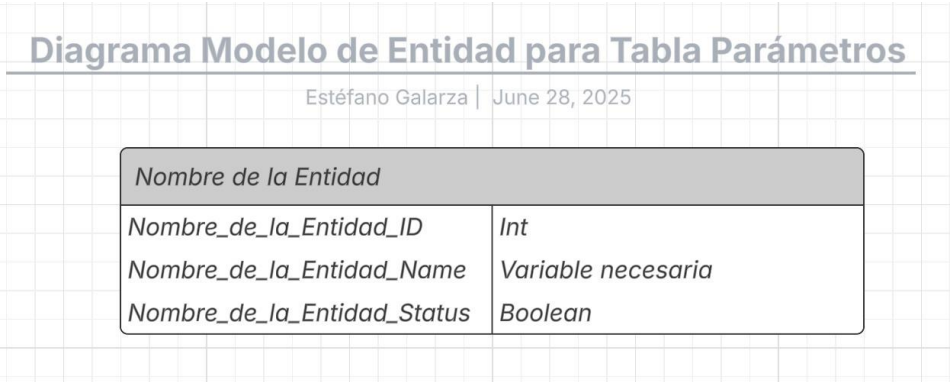
Field	Type	Null	Key	Default	Extra
Record_ID	int	NO	PRI	NULL	auto_increment
Activity_ID	int	YES	MUL	NULL	
Activity_Record_Type	varchar(50)	YES		NULL	
Activity_Record_Recorded_Time	datetime	YES		NULL	
Activity_Record_Latitude	decimal(10,8)	YES		NULL	
Activity_Record_Longitude	decimal(11,8)	YES		NULL	
Activity_Record_On_Time	tinyint(1)	YES		NULL	
Activity_Record_Observation	text	YES		NULL	

Nota: Elaboración propia. Entidad Activity_Records en la base de datos.

Para las entidades de parámetros, existen 36 entidades solicitadas por los consultorios jurídicos gratuitos de la PUCE, pero todas se basan en una misma plantilla como se observa en la **Ilustración 23**.

- Nombre_De_La_Entidad_ID, será un número entero autoincrementable.
- Nombre_De_La_Entidad_Name, variable para almacenar el parámetro.
- Nombre_De_La_Entidad_Status, en este atributo se guarda un valor booleano (0 / 1), donde se establece el estado el parámetro. Para esto se utiliza un borrado lógico, el parámetro solo cambia de estado sin desaparecer de la base de datos.

ILUSTRACIÓN 23 PLANTILLA TABLA DE PARÁMETROS



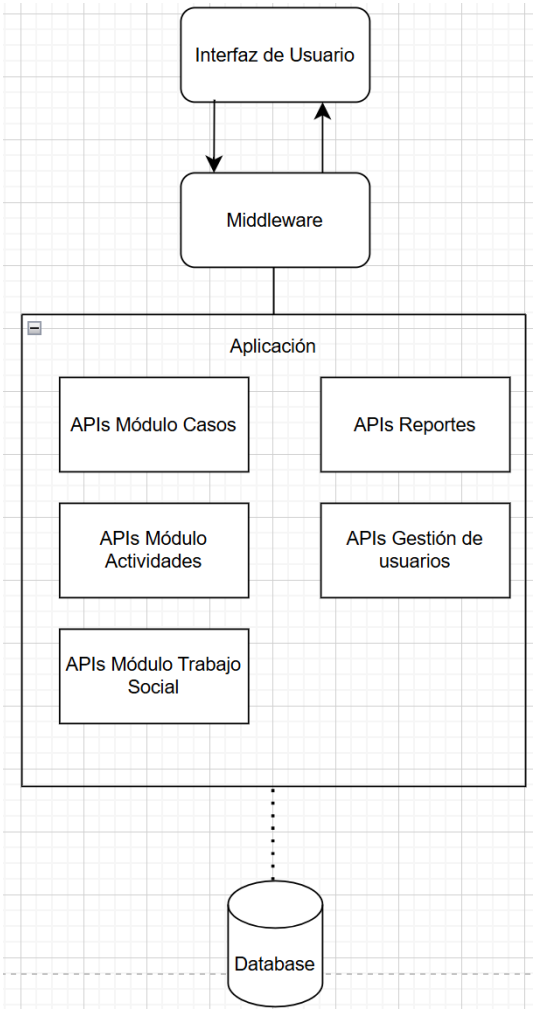
Nota: Elaboración propia. Plantilla "Tabla de parámetros" en la base de datos.

3.4.2 Estructura del proyecto BackEnd (API-RESTful - MVC)

El sistema Balanza Web para los Consultorios Jurídicos gratuitos de la PUCE se divide en 3 partes, Front End, Back End y Modelo de base de datos. La estructura del programa se basa en la comunicación de estas partes a través de APIs con sus respectivas rutas.

La siguiente imagen, **Ilustración 24**, se puede observar de manera clara la estructura y la conexión entre las diferentes partes.

ILUSTRACIÓN 24 ESTRUCTURA RELACIÓN FRONTEND, BACKEND Y BASE DE DATOS



Nota: Elaboración propia. Diagrama de estructura sistema, relación tres partes Frontend, Backend y base de datos.

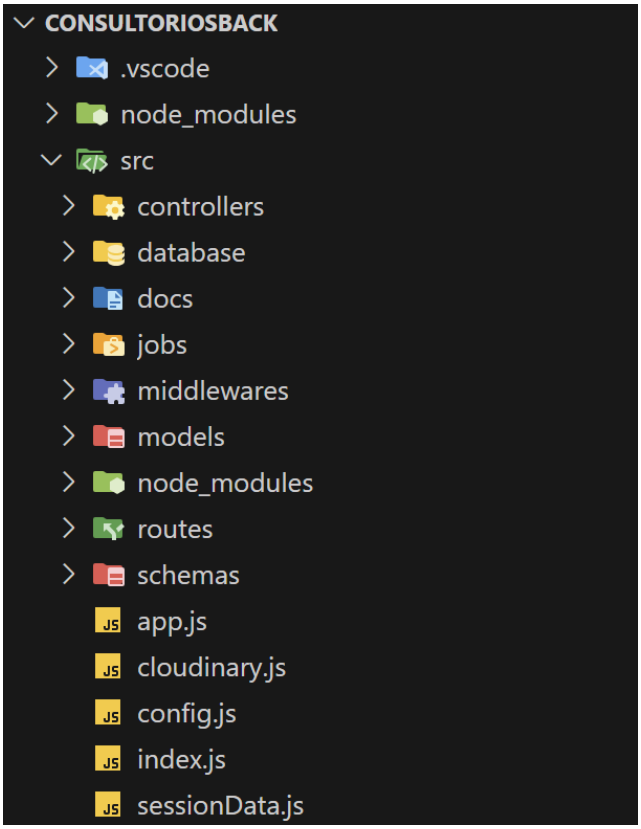
La parte BackEnd del sistema se basa en la arquitectura API RESTful el cual es un estilo de diseño para desarrollar servicios web ligeros y escalables. Sigue principios que permiten que el backend y el frontend se comuniquen a través de peticiones HTTP estándar (GET, POST, PUT, DELETE).

El proyecto también tiene una estructura MVC en el que el modelo es lo que contiene todo con referente a las entidades de la base de datos junto con métodos para realizar GET, POST, PUT Y DELETE. Por otra parte, están los controladores, en el

que se va a hacer referencia a los métodos de cada entidad. Y por utlimo la vista, que en este caso son las rutas para que el FrontEnd pueda acceder a los métodos y a las debidas entidades mediante las rutas.

La estructura de los modelos, rutas y controladores se pueden observar en la **Ilustración 25** de la siguiente manera:

ILUSTRACIÓN 25 ESTRUCTURA DEL PROYECTO BACKEND



Nota: Elaboración propia. Estructura del BackEnd .

- En schemas se encuentran todas las entidades de la base de datos para poder mapearla con sequelize.
- En models, se encuentran los métodos necesarios para gestionar cada entidad (GET, POST, PUT, DELETE).
- En controladores, se encuentran las llamadas a estos métodos en los modelos.

- En routes, se encuentran todas las rutas para darle acceso al FrontEnd a los métodos respectivos de cada módulo.
- En middlewares, se encuentra código necesario para autenticar aspectos del sistema y poder comunicar al FrontEnd con el BackEnd. También se tiene un archivo para poder subir archivos PDF a la base de datos, subir imágenes para el avatar del perfil, autenticación de token sobre la sesión activa en el momento y un archivo para determinar que pantallas se van a mostrar a un usuario dependiendo de su rol y permisos en el sistema.
- En docs, se encuentran la plantilla para crear la ficha técnica del caso y también se encuentra la fuente de texto necesaria solicitad por la Dirección de informática de la PUCE
- En jobs, se encuentran métodos para gestionar las asistencias de los usuarios internos con rol de estudiantes
- En database, en este apartado se encuentra lo necesario para conectarse a la base de datos de manera local para realizar las pruebas necesarias. Este apartado contiene, host, user, password y database.

CAPÍTULO IV: DESARROLLO DE LA APLICACIÓN

4.1 Preparación del entorno de desarrollo

4.1.1 Estándares de codificación

Para preparar el entorno de desarrollo para el sistema web, fue de suma importancia tomar en cuenta los alineamientos del entorno tecnológico de la PUCE. Inicialmente herramientas que se conocían podrían ser una opción viable, pero, aun así, esto podía ser problemático al centro de informática.

Para el desarrollo de esta aplicación, se tuvo que evaluar en qué lenguajes de programación nos era más familiar y también el motor de base de datos que podríamos usar con facilidad. Pero después, la propia dirección de informática de la PUCE, al trabajar en conjunto con esta área de TI, nos impuso herramientas y lenguaje de programación que ya utilizaban para este tipo de desarrollo.

En este caso se nos impuso el uso de vue.js para el desarrollo FrontEnd, y JavaScript para el desarrollo del BackEnd. Para el uso de estos se debió tener listo un IDE, en este caso se optó por Visual Studio Code, para ejecutar el back end con JavaScript, se utilizó Node.js, como motor de base de datos se utilizó MySQL 8.0 y para el control de versiones se utilizaron dos repositorios en GitHub.

Al mismo tiempo también se tomaron en cuenta varios parámetros para estar acorde con los estándares de codificación de la Dirección de informática de la PUCE. Para esto, se mencionaron: Nombres de variables y entidades en idioma Inglés, y también, funciones en camelCase. Para otras convenciones se tomó en cuenta la separación lógica de carpetas, models, controllers, routes, middlewares, schemas, docs, Jobs y database.

4.2 Implementación de la funcionalidad

4.2.1 BackEnd

El Back End se desarrolló usando la arquitectura API Rest que nos ayuda a tener una conexión entre el Front End, Back End y modelo de datos de forma eficiente y precisa vía solicitudes HTTP para métodos como GET, POST, PUT y DELETE.

En toda la implementación del Back End se incluyó:

- ORM para poder mapear la base de datos directamente desde el BackEnd. Para ello se utilizó Sequelize para un fácil mapeo hacia la base de datos de MySQL.

- Multer, con ello se pudo plantear los debidos parámetros para la subida de archivos, tanto PDF como imágenes.
- Una automática gestión de Auditoria en las interacciones más importantes del sistema, en este caso se registra cuando se crea, edita, elimina un elemento del sistema que se encuentre en la base de datos. Se establecieron casos de uso específicos para el módulo de auditoría, como:
    - CU02: Registro general de acciones.
    - CU03: Auditoría de creación de registros.
    - CU04: Auditoria de actualizaciones.
    - CU05: Auditoria de eliminaciones.
- Se tiene las plantillas para la creación de reportes, se tiene la ficha técnica que se utiliza de manera oficial en los Consultorios jurídicos gratuitos de la PUCE.

Por último, para tener un código organizado, limpio y mantenible se optó por la separación lógica de carpetas, models, controllers, routes, middlewares, schemas, docs, Jobs y database.

4.3 Plan de pruebas

4.3.1 Tipos de pruebas

Probar software es el proceso en donde se evalúa y verifica que el sistema que está en proceso de desarrollo cumpla con cualquier criterio que se planteó al inicio del proyecto. Realizar pruebas precisas y exhaustivas ayuda a la prevención de errores y mejora el rendimiento de este.

Existen dos apartados, las pruebas manuales y las pruebas automatizadas. Las pruebas manuales son realizadas por una persona interactuando directamente con el sistema, dando click navegando por la aplicación. Estas pruebas son bastante costosas, tanto económicamente como en tiempo. Por otro lado, se encuentra las pruebas automatizadas las cuales son realizadas por una máquina corriendo un script pre

hecho, estas pruebas automatizadas pueden probar desde algo muy pequeño hasta algo grande. Estas pruebas son más potentes que las manuales, pero todo depende de qué tan bien se realice el script para probar.

Dentro de nuestro sistema, se optó por realizar pruebas unitarias con la metodología Prototipado, aunque existen más tipos de pruebas tales como:

- Pruebas unitarias
- Pruebas de integración
- Pruebas funcionales
- Pruebas de extremo a extremo
- Pruebas de aceptación
- Pruebas de rendimiento
- Pruebas de humo

4.3.1.1 Pruebas de unidad en Auditoría

Las pruebas de unidad es un proceso que el que se centra en realizar pruebas a un sistema desde lo más pequeño del código. El testeo del software ayuda a asegurar un código limpio y como se puede integrar con las demás partes. Como buena práctica, la prueba de unidad es bastante sólida, primero realizamos el sistema por partes muy pequeñas y así podemos probar cada una de manera correcta, y en el caso de que haya algún error, se sabrá exactamente en qué parte se encuentra.

También se pueden realizar pruebas unitarias de forma automatizada, esto ayuda a que nosotros los programadores nos concentremos en el código y no tanto en realizar las pruebas, claro que igual se deben realizar las mismas, pero esto lleva tiempo.

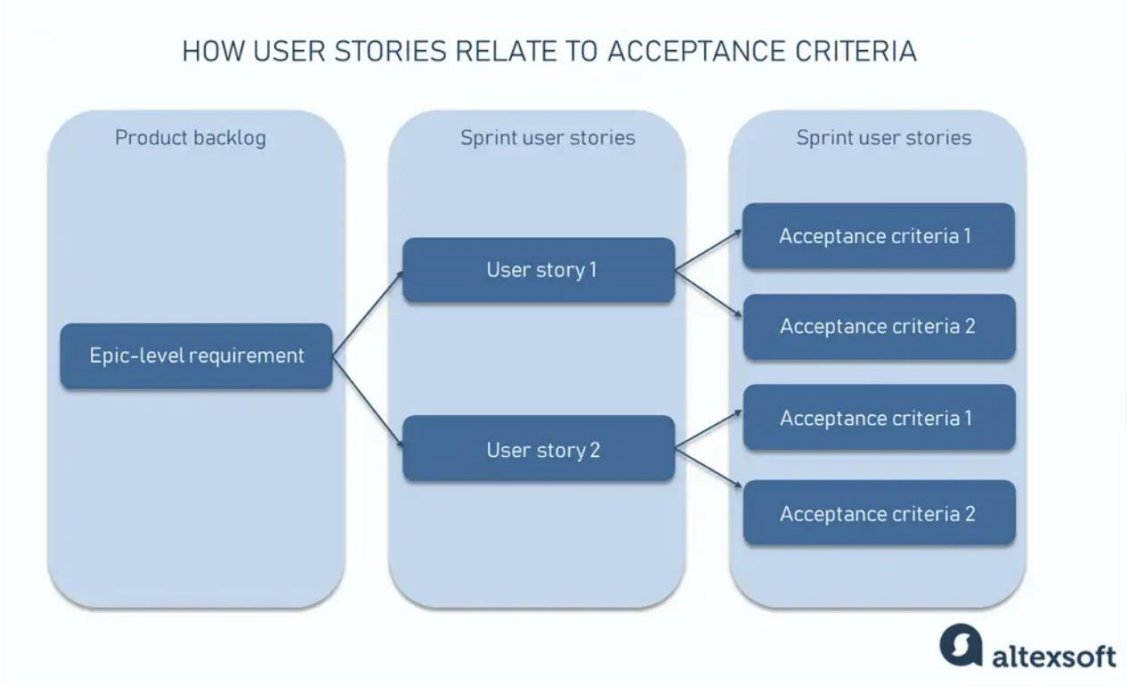
Para el módulo de auditoría, se realizaron pruebas unitarias de manera exhaustiva y precisa. Al inicio del proyecto se utilizaron herramientas como Postman y

ThunderClient ya que solo se tenía acceso al BackEnd, pero al momento de integrar las diferentes partes, FrontEnd y BackEnd, se pudieron realizar pruebas de Auditoria directamente en la interfaz de usuario.

4.3.2 Criterios de aceptación

Los criterios de aceptación son condiciones propuestas por el equipo, tomando en cuenta requisitos funcionales y no funcionales para que el producto final sea aceptado por el usuario receptor del sistema. Un planteamiento correcto de estos criterios ayuda a evitar confusiones y resultados inesperados y, como se puede observar en la **Ilustración 26**, se basan plenamente en historias de usuario, determinando el qué es lo que el sistema necesita.

ILUSTRACIÓN 26 HISTORIAS DE USUARIO – CRITERIOS DE ACEPTACIÓN



Nota: Elaborado por altexsoft. Diagrama donde se ve la relación que se debe tener para realizar los criterios de aceptación basándose en las historias de usuario.

Para las pruebas del módulo de Auditoría se plantearon varios criterios de aceptación.

- Cada acción critica en el sistema por parte del usuario interno, ya sea CREATE, UPDATE o DELETE de registros, deben generar el correspondiente reporte o historial en la entidad de Audits en la base de datos.
- Cada registro nuevo en la entidad Audits, deben tener presente el Internal_ID del quien hizo la acción, la acción realizada, la entidad afectada de la base de datos y la fecha y hora en la que se realizó dicha acción.
- Si el token es invalido, la acción no se registrará en la base de datos, el usuario podría haber excedido el tiempo límite de la sesión sin realizar ninguna acción por lo que el token se invalida y la tabla Audits ya no registrará acciones del usuario.
- La entidad de Auditoria no debe ser accesible para cualquier usuario, por ello no existe ninguna interfaz de usuario para poder consultar, editar o eliminar alguno de los registros de esta tabla. La única forma de poder ver estos registros es directamente en la base de datos en donde se necesita permisos especiales directamente con el área de base de datos en la Dirección de informática de la PUCE.

4.3.3 Casos de prueba

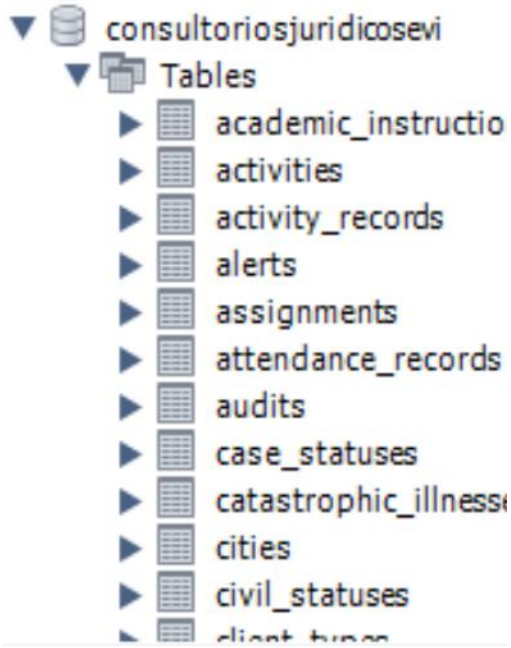
Para probar la funcionalidad del registro automático de auditoría en cada interacción con la base de datos a través del sistema, se tuvo que realizar un flujo completo de trabajo sobre el registro de casos en los consultorios jurídicos gratuitos de la PUCE. Se tomo capturas de pantalla tanto de la interacción del sistema, como capturas de pantalla sobre la entidad Audits en la base de datos para observar si se registraban dichas acciones de forma correcta.

Para realizar las pruebas primero se creó una base de datos de forma local para un entorno de pruebas, “consultoriosjurídicosevi”, totalmente vacía (**Ilustración 27**). Sobre esta base de datos se mapea las entidades, sé observa en la **Ilustración 28,** que se encuentran el BackEnd y se procede con la prueba. También se creó un script el cual inserta valores propuestos por los Consultorios jurídicos gratuitos de la PUCE sobre la tabla de parámetros.

ILUSTRACIÓN 27 BASE DE DATOS DE PRUEBA VACÍA

Nota: Elaboración propia. Esta es la imagen de la base de datos vacía, antes de mapear las entidades del BackEnd.

ILUSTRACIÓN 28 BASE DE DATOS CON ENTIDADES MAPEADAS



Nota: Elaboración propia. Esta es la imagen de la base de datos con las entidades ya ingresadas mediante el BackEnd con el ORM Sequelize.

El flujo de creación de caso en los que se deberá registrar en Auditoria es el siguiente:

- Primero, como se puede observar en la **Ilustración 29,** verificamos que la entidad Audits se encuentra totalmente vacía. Pero para realizar las pruebas se debe crear un usuario con rol de abogado y un usuario con el rol de estudiante, se puede ver

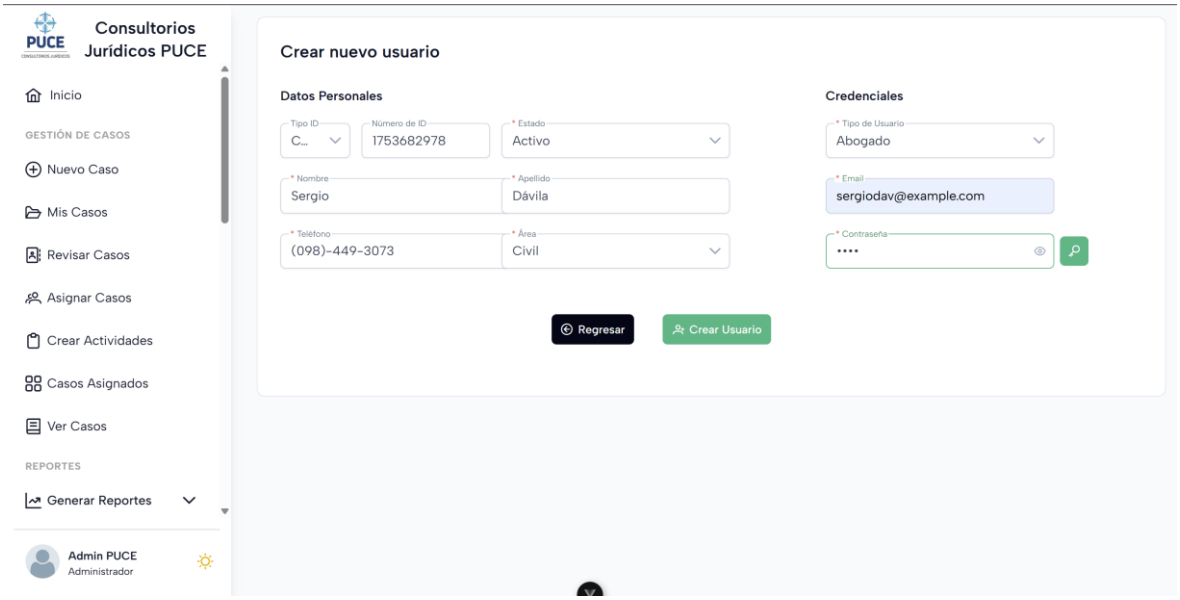
en la **Ilustración 30 e Ilustración 31,** que sean de la misma área, esto se va a ver reflejado en la tabla Audits como se ve en la **Ilustración 32** con el Internal_Id del usuario interno con rol de administrador creado por defecto con cedula 0000000000.

Ilustración 29 Entidad Audits vacía

Id_Audit	Internal_ID	Audit_Accion	Audit_Tabla	Audit_Descripcion
NULL	NULL	NULL	NULL	NULL

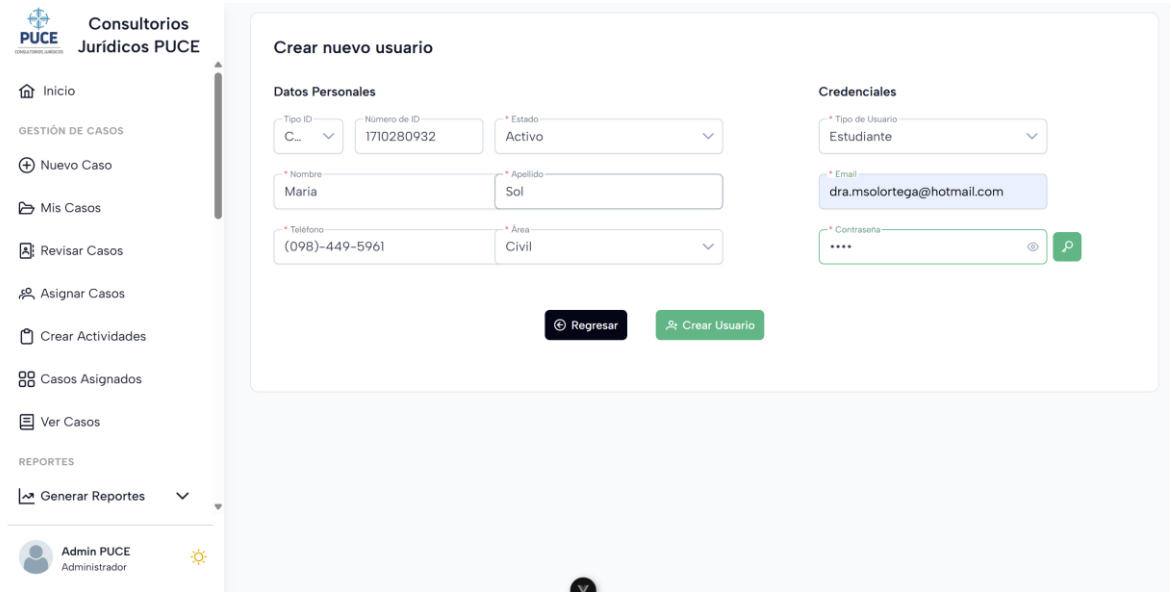
Nota: Elaboración propia. Entidad Audits totalmente vacía antes de realizar pruebas para que no existan confusiones.

Ilustración 30 Creación de abogado interfaz de usuario



Nota: Elaboración propia. Creación de usuario interno con rol de "Abogado" mediante la interfaz de usuario.

Ilustración 31 Creación de estudiante interfaz de usuario



Nota: Elaboración propia. Creación de usuario interno con rol de "Estudiante" mediante la interfaz de usuario.

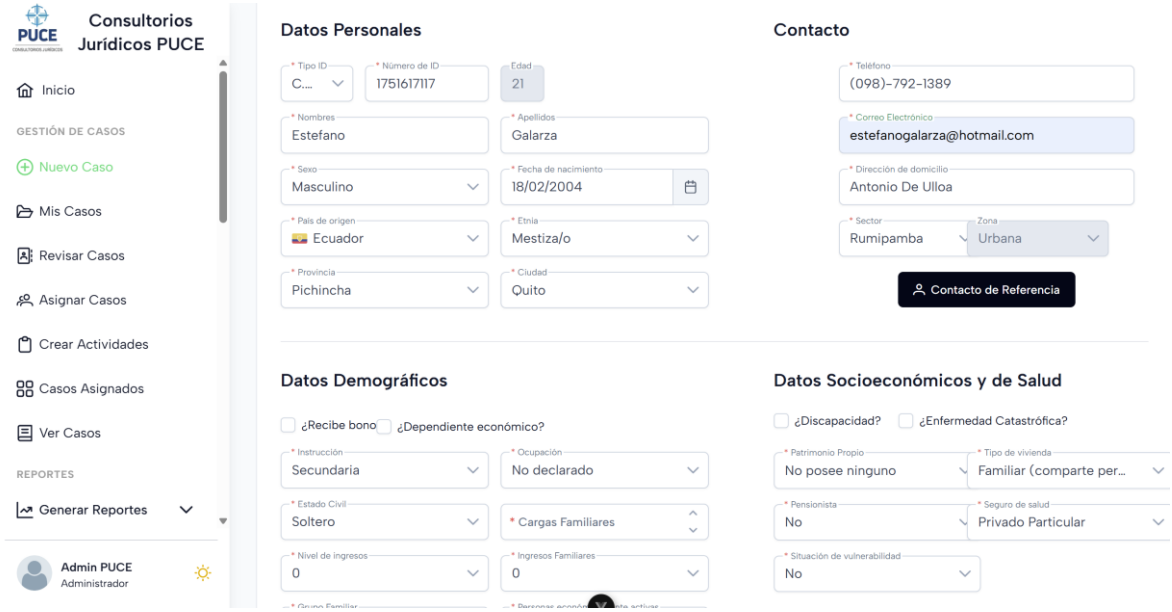
ILUSTRACIÓN 32 ENTIDAD AUDITS CON REGISTROS DE CREACIÓN DE USUARIOS INTERNOS

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
NULL	NULL	NULL	NULL	NULL	NULL

Nota: Elaboración propia. Entidad Audits con registros de creación de usuarios internos con rol de "Abogado" y "Estudiante"

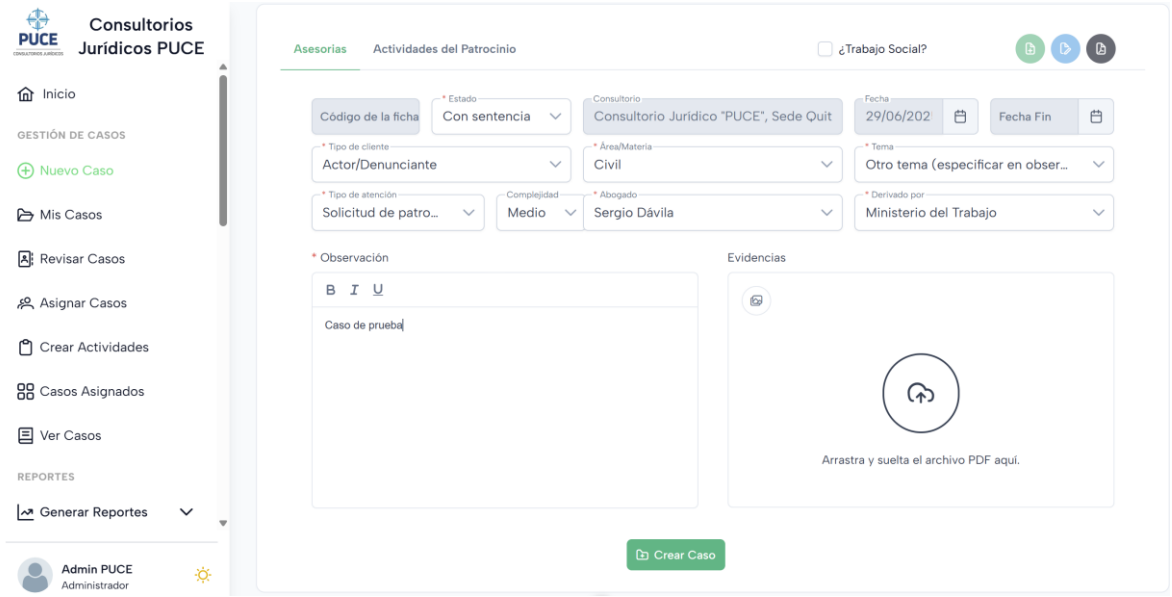
- Creación de caso, para ello se debe contar con un abogado y estudiantes de la misma área para que el abogado reciba el caso creado y pueda delegarlo a algún estudiante de su área. La creación de casos altera tres entidades, Users, Initial_Consultations y Evidences, como se puede ver en la **Ilustración 33, Ilustración 34** y el resultado en Auditoria en la **Ilustración 35**.

Ilustración 33 Creación de caso interfaz de usuario, tabla User



Nota: Elaboración propia. Creación de usuario externo en la interfaz de usuario”

ILUSTRACIÓN 34 CREACIÓN DE CASO EN INTERFAZ DE USUARIO, INITIAL_CONSULTATIONS Y EVIDENCES



Nota: Elaboración propia. Creación de consulta inicial en la interfaz de usuario junto con su evidencia.

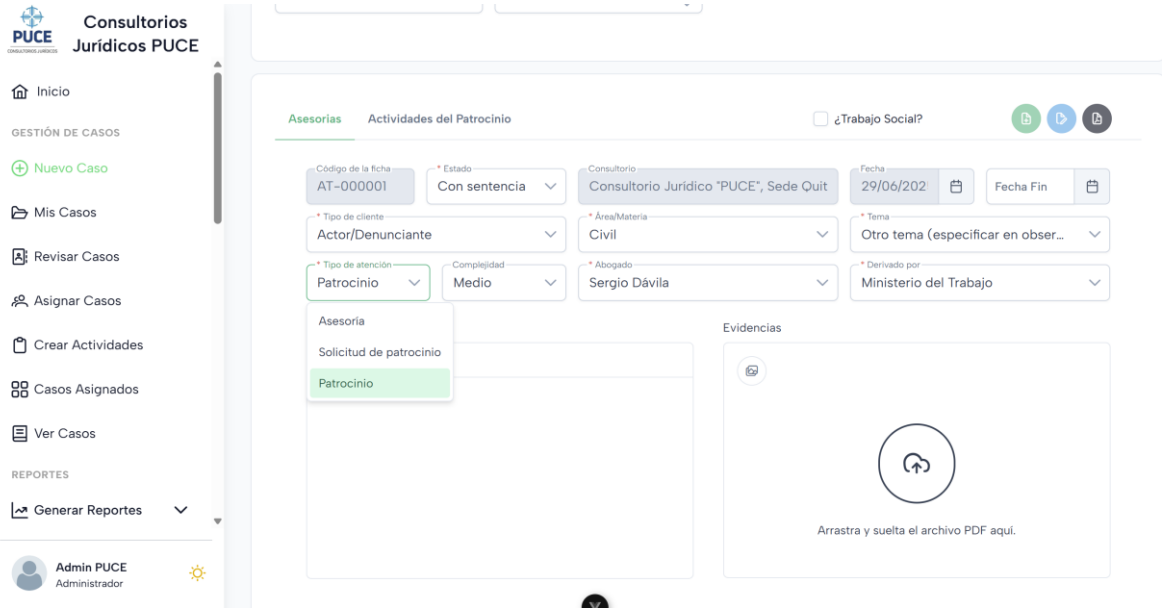
ILUSTRACIÓN 35 REGISTRO DE CREACIÓN EN LA ENTIDAD AUDITS

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. Registros en la entidad Audits para comprobar la creación de dichos elementos.

- Se procede a revisar el caso, y en caso de que el abogado crea qué si es viable continuarlo, el abogado mismo tiene que editar el caso y colocarlo como "Patrocinio" (**Ilustración 36 e Ilustración 37**).

Ilustración 36 Cambio de caso a patrocinio



Nota: Elaboración propia. Edición de caso a Patrocinio en la interfaz de usuario una vez revisado por el abogado del área.

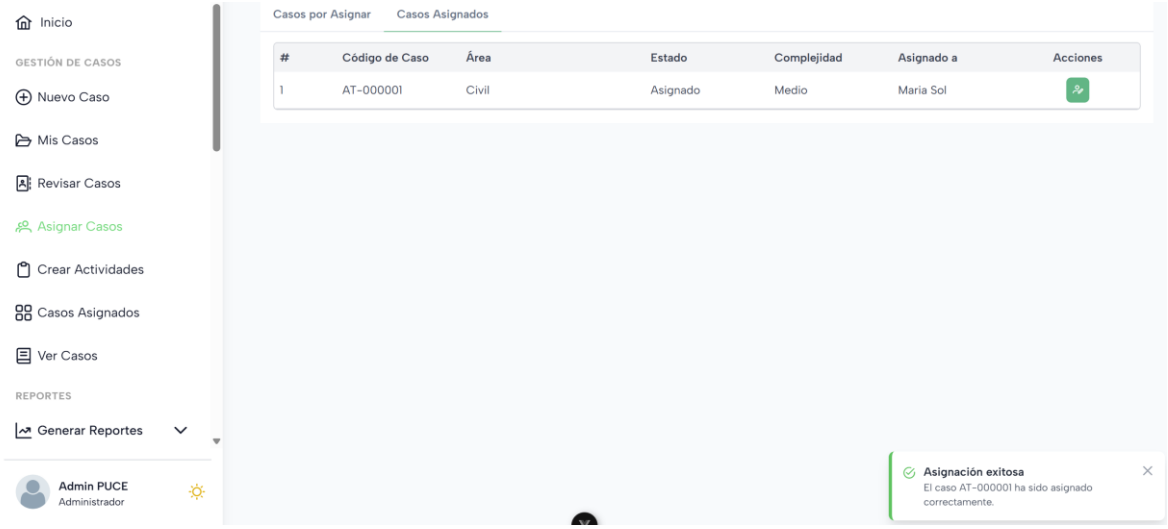
ILUSTRACIÓN 37 REGISTRO DE ACTUALIZACIÓN DE CASO EN AUDITS

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
6	0000000000	UPDATE	Initial_Consultations	El usuario interno 0000000000 actualizó la consulta inicial AT-000001	2025-06-29 23:39:00
NULL	NULL	NULL	NULL	NULL	NULL

Nota: Elaboración propia. Registro del cambio de Tipo de atención a Patrocinio en la entidad Audits de la base de datos.

- Una vez el caso esté en patrocinio, el abogado puede delegarlo a algún estudiante de su área, como se ve en la **Ilustración 38** y su debido registro en Audits en la **Ilustración 39**.

ILUSTRACIÓN 38 DELEGACIÓN DE CASO INTERFAZ DE USUARIO



Nota: Elaboración propia. Asignación de casos en la interfaz de usuario una vez cambiado a Patrocinio.

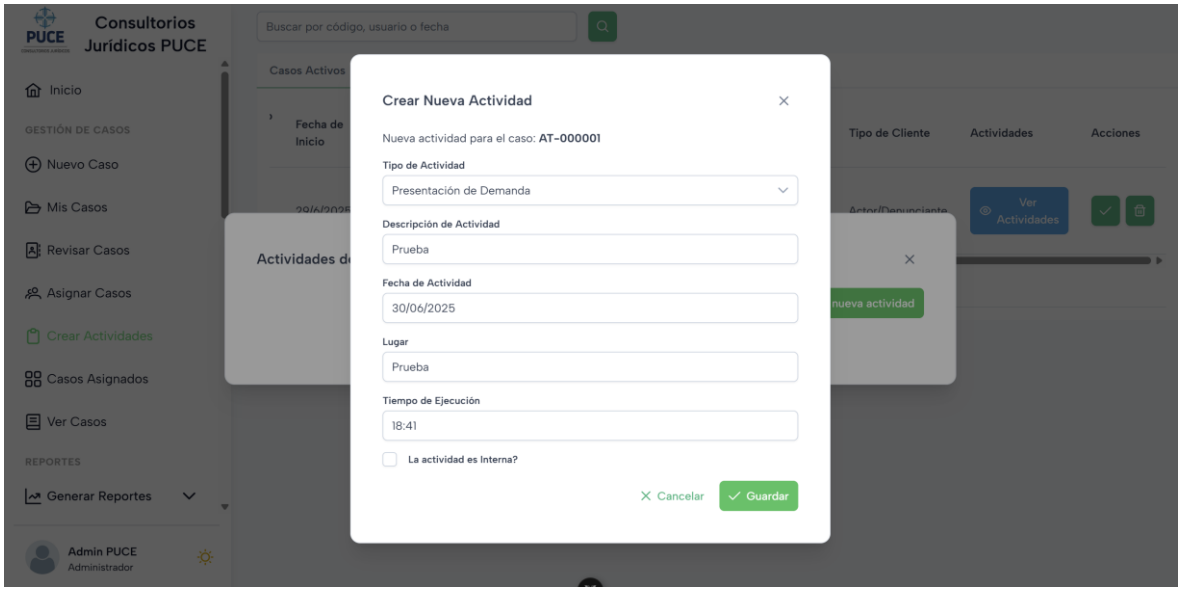
ILUSTRACIÓN 39 REGISTRO DE ASIGNACIÓN DE CASO AUDITS

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
6	0000000000	UPDATE	Initial_Consultations	El usuario interno 0000000000 actualizó la consulta inicial AT-000001	2025-06-29 23:39:00
7	0000000000	INSERT	Assignment	El usuario interno 0000000000 creó la asignación con ID 1	2025-06-29 23:39:54
8	0000000000	UPDATE	Initial_Consultations	El usuario interno 0000000000 actualizó la consulta inicial AT-000001	2025-06-29 23:39:54

Nota: Elaboración propia. Registro de la asignación del caso en la entidad Audits de la base de datos.

- Al delegarlo, el abogado puede crear actividades sobre el caso, actividades que el estudiante deberá cumplir en el tiempo establecido como se ve en la **Ilustración 40** y su debido registro en Audits en la **Ilustración 41**.

ILUSTRACIÓN 40 CREACIÓN DE ACTIVIDADES INTERFAZ DE USUARIO



Nota: Elaboración propia. Creación de actividades sobre el caso en la interfaz de usuario.

ILUSTRACIÓN 41 REGISTRO DE CREACIÓN DE ACTIVIDADES EN 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
6	0000000000	UPDATE	Initial_Consultations	El usuario interno 0000000000 actualizó la consulta inicial AT-000001	2025-06-29 23:39:00
7	0000000000	INSERT	Assignment	El usuario interno 0000000000 creó la asignación con ID 1	2025-06-29 23:39:54
8	0000000000	UPDATE	Initial_Consultations	El usuario interno 0000000000 actualizó la consulta inicial AT-000001	2025-06-29 23:39:54
9	0000000000	INSERT	Activity	El usuario interno 0000000000 creó la actividad con ID 1	2025-06-29 23:41:57
NULL	NULL	NULL	NULL	NULL	NULL

Nota: Elaboración propia. Registro de la creación de actividades en la entidad Audits de la base de datos.

Fuera del flujo se encuentra aspectos que igual cuentan con registro automático de auditoria como gestión de usuarios y gestión de parámetros.

4.4 Documentación del código

4.4.1 Auditoría en FrontEnd

El FrontEnd se responsabiliza en una adecuada interacción que tiene un usuario con el sistema. Sin embargo, el módulo de auditoria no es accesible ni visible para cualquier usuario, solo se lo puede acceder mediante la base de datos con usuario y contraseña en modo Admin.

En el FrontEnd se importan el store de autenticación en conjunto con axios, el cual nos ayuda con las peticiones HTTP al BackEnd. También se declara la constante authStore para almacenar los datos del usuario en la sesión actual, véase en el **Anexo 1,** con ello se puede extraer el Internal_Id y enviarlo al BackEnd para realizar la auditoria, véase en el **Anexo 2**.

4.4.2 Auditoría en BackEnd

El BackEnd incluye schemas definidos para ser mapeados en la base de datos mediante Sequelize, schemas que representan todas las entidades necesarias del sistema. Entre estas

entidades se encuentra AuditModel, véase en el **Anexo 3,** junto con su controlador AuditController en el **Anexo 4**, el cual contiene los métodos necesarios para poder registrar cualquier acción de CRUD de las diferentes entidades.

La entidad Audit, necesita ser llamada en cada una de las demás entidades para registrar correctamente el método que se realizó en la entidad correspondiente. Por lo que en cada API para realizar CRUD en todas las entidades, se encontraba la función de Audit para poder registrar la debida auditoria en la base de datos. Véase la función de crear auditoría en el **Anexo 5**

CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Al finalizar el proyecto se analizó lo el producto final, por ello se realizó conclusiones sobre todo el trabajo y el tiempo empleado para cumplir con el cronograma.

- La aplicación Balanza Web representa una mejora significativa e importante para automatizar los procesos en los Consultorios Jurídicos de la PUCE. Aunque al momento de realizar la implementación y cambiar su sistema anterior, que ya estaba un poco anticuado, por el nuestro, debemos tener mucho cuidado a la resistencia al cambio. Estudiantes que ya se encontraban con el anterior sistema pueden no adaptarse rápido o mostrar resistencia al nuevo sistema.

- Al integrar un módulo de auditoria en el sistema, significa una ventaja contra el anterior sistema. Al tener este módulo, se puede realizar de mejor manera un monitoreo de procesos de los Consultorios Jurídicos de la PUCE.

- El desarrollo del BackEnd usando JavaScript con Node.js y el ORM Sequelize, nos facilitó la integración con la base de datos en MySQL. Al momento del desarrollo se tuvieron que realizar muchos cambios a la base de datos, y con Sequelize, nos resultó más sencillo solo cambiar el código y que este se vea reflejado de forma automática en la base de datos

- Realizar un correcto plan de pruebas ayudó a validar y verificar el funcionamiento del módulo de auditoría. Se pudo ver paso a paso cada proceso del flujo de casos en el sistema, como el crear casos, gestión de usuarios, asignación y creación de actividades, y gestión de parámetros del sistema.

- A pesar de todo lo realizado del desarrollo del proyecto, el sistema aún no está terminado en su totalidad, pero está bastante bien estructurado para un siguiente grupo pueda tomar este proyecto y pulirlo para que este 100% funcional.

5.2 Recomendaciones

Al finalizar el proyecto existieron varios puntos a mejorar y a tomar en cuenta en el caso de realizar proyectos de desarrollo en el futuro, para ello se realizó recomendaciones.

- Se recomienda para un futuro, implementar una interfaz de usuario para tener interacción con auditoria directamente en el sistema y no por medio de la base de datos. Claro que, al implementar una interfaz, también se debe adicionar algún rol para que solo usuarios con la respectiva autorización puedan acceder a esta interfaz.

- Es importante realizar pruebas de estrés y de seguridad para evaluar el rendimiento de los diferentes módulos, en este caso, el módulo de auditoría. Realizar pruebas de estrés, significa evaluar el rendimiento mientras se realizan muchas cargas de interacciones en el sistema.

- Se sugiere mantener el sistema propiamente documentado periódicamente. Esto facilitará el entendimiento, mantenibilidad y escalabilidad del sistema para futuros cambios. Tener documentado cada función hasta como sé debe tener el entorno de desarrollo para que no exista ningún problema.

- Realizar una apropiada capacitación a los usuarios internos para que estos puedan utilizar el sistema de manera apropiada y con responsabilidad. Aquí se resalta el módulo de Auditoria para que exista transparencia y rendición de cuentas dentro de los Consultorios Jurídicos de la PUCE.

Anexos

ANEXO 1 IMPORTACIONES FRONTEND AUDITORIA

```
import { useAuthStore } from "@/stores/auth";
import axios from "axios";

const authStore = useAuthStore();
```

Nota: Elaboración propia. Se importan librerías como axios y se importa el store de la sesión actual.

ANEXO 2 MÉTODO FRONTEND

```
await axios.post(urlMap[selectedTableKey.value], selectedRecord.value, {
  headers: {
    "internal-id": authStore.user?.id,
  },
});
```

Nota: Elaboración propia. Método axios se hace una solicitud POST al backend para crear el nuevo registro, enviando los datos (selectedRecord.value) y el internal-id del usuario.

ANEXO 3 MODELO AUDITMODEL

```
static async registerAudit(internalId, accion, tabla, descripcion) {
    try {
        // Verificar que el Internal_ID existe en la tabla internal_users
        const internalUser = await InternalUser.findOne({ where: { Internal_ID: internalId } });
        if (!internalUser) {
            throw new Error(`El Internal_ID ${internalId} no existe en la tabla internal_users`);
        }

        return await Audit.create({
            Internal_ID: internalId,
            Audit_Accion: accion,
            Audit_Tabla: tabla,
            Audit_Descripcion: descripcion
        });
    } catch (error) {
        console.error(`Error al registrar auditoria: ${error.message}`);
    }
}
```

Nota: Elaboración propia. Modelo AuditModel, recibe el Internal_Id para poder realizar el registro en la tabla Audits

ANEXO 4 CONTROLADOR AUDITCONTROLLER

```
static async registerAudit(req, res) {
    try {
        const { Internal_ID, Audit_Accion, Audit_Tabla, Audit_Descripcion } = req.body;

        if (!Internal_ID || !Audit_Accion || !Audit_Tabla || !Audit_Descripcion) {
            return res.status(400).json({ error: "Todos los campos son obligatorios" });
        }

        const newAudit = await AuditModel.registerAudit(Internal_ID, Audit_Accion, Audit_Tabla, Audit_Descripcion);
        res.status(201).json({ message: "Auditoría registrada", data: newAudit });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
}
```

Nota: Elaboración propia. Modelo AuditController, recibe el Internal_Id y los demás parámetros para poder realizar el registro en la tabla Audits.

ANEXO 5 MÉTODO REQUERIDO

```
const internalId = internalUser || getUserId();
// ✅ Registrar en Audit quién creó el usuario
await AuditModel.registerAudit(
    internalId,
    "INSERT",
    "User",
    `El usuario interno ${internalId} creó al usuario ${newUser.User_ID}`, // Use newUser.User_ID
    { transaction: t }
);
```

Nota: Elaboración propia. Método en todos los métodos CRUD de todos los modelos. Recibe el Internal_Id para poder realizar el registro en la tabla Audits con AuditModel y AuditController.

Bibliografía

- Reactive Programming. (s.f.). *Arquitectura Cliente-Servidor*. Recuperado de: https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/cliente-servidor

- Santander Universidades. (2020, 21 de diciembre). *Metodologías de desarrollo de software: ¿qué son?*. Recuperado de: https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html

- Pure Storage. (s.f.). *¿Qué es MySQL? ¿Y por qué sigue siendo la base de datos más popular?*. Recuperado de: https://www.purestorage.com/es/knowledge/what-is-mysql.html

- Kinsta. (2025, 6 de marzo). *¿Qué es JavaScript? Un repaso al lenguaje de programación de scripts más popular de la web*. Recuperado de: https://kinsta.com/es/base-de-conocimiento/que-es-javascript/

- Arsys. (s.f.). *¿Qué es Visual Studio Code y cuáles son sus ventajas?*. Recuperado de: https://www.arsys.es/blog/que-es-visual-studio-code-y-cuales-son-sus-ventajas

- Kinsta. (2025, 6 de marzo). *Qué es Node.js y por qué deberías usarlo.* Recuperado de: https://kinsta.com/es/base-de-conocimiento/que-es-node-js/

- IBM. (s.f.). *JSON Web Token (JWT)*. Recuperado de: https://www.ibm.com/docs/es/cics-ts/6.x?topic=cics-json-web-token-jwt

- Webempresa. (s.f.). *Bcrypt: ¿Qué es y para qué funciona?*. Recuperado de: https://www.webempresa.com/blog/bcrypt-que-es-y-para-que-funciona.html

- IronPDF. (s.f.). *Multer Node.js*. Recuperado de: https://ironpdf.com/es/nodejs/blog/node-help/multer-nodejs/#trial-license

- IronPDF. (s.f.). *Sequelize Node.js*. Recuperado de: https://ironpdf.com/es/nodejs/blog/node-help/sequelize-node-js/

- Unity. (s.f.). *¿Qué es el control de versiones?.* Recuperado de: https://unity.com/es/topics/what-is-version-control

- Microsoft. (s.f.). *Fundamentos de Git y GitHub*. Recuperado de: https://learn.microsoft.com/es-es/contribute/content/git-github-fundamentals

- Cloudflare. (s.f.). *¿Qué es la seguridad de API?*. Recuperado de: https://www.cloudflare.com/es-es/learning/security/api/what-is-api-security/

- GetKonvex. (s.f.). *Manejo de errores en API: mejores prácticas para implementar*. Recuperado de: https://www.linkedin.com/pulse/manejo-de-errores-api-mejores-pr%C3%A1cticas-implementar-getkonvex-nu4qe/

- AWS. (s.f). "*¿Qué son las pruebas unitarias?*". Recuperado de: https://aws.amazon.com/es/what-is/unit-testing/

- Altexsoft. (2023). "*Acceptance criteria for User stories in Agile*". Recuperado de: https://www.altexsoft.com/blog/acceptance-criteria-purposes-formats-and-best-practices/

- Sten Pittet. (s.f). "*Los distintos tipos de pruebas de software*". Recuperado de: https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing

- IBM. (2021). "*Qué son las pruebas de software?*". Recuperado de: https://www.ibm.com/es-es/think/topics/software-testing