**Trabajo Practico: Análisis**

**Colegio superior Nº47 “Florentino Ameghino”**

**Carrera: Técnico Superior en Análisis Funcional de Sistemas Informáticos**

**Materia: Practica Profesionalizante**

**Integrantes: Román Valle – Tobías Lazos – Mateo Molina**

**Profesor: Leandro Gómez**

**Índice**

**Modulo I 2**

**Modulo II 8**

**Modulo III 43**

**Modulo IV 71**

**Modulo V 85**

**Modulo I**

**Pautas Generales y de Organización:**

**Planificación de Actividades.**

**Evaluación de Oportunidades.**

**Oportunidad 1**

**Realizar un sistema de control de medicamentos y de revisión médica para una residencia de adultos mayores.**

**Análisis:**

* **Mercado:** Específico, pero con un alto impacto en la calidad de vida de los residentes.
* **Competencia:** Baja, ya que este tipo de software suele ser personalizado.
* **Tecnología:** Disponible y accesible.
* **Recursos internos:** Dependerá de la capacidad técnica de la residencia.
* **Riesgo:** Medio.

**Beneficios:**

* Mejor control del inventario de medicamentos.
* Mayor eficiencia en la revisión médica.
* Reducción de errores en la administración de medicamentos.
* Mejora en la calidad de vida de los residentes.

**Oportunidad 2**

**Realizar un sistema de recetas para una organización sin fines de lucro que tiene como objetivo combatir la desnutrición infantil.**

**Análisis:**

* **Mercado:** Amplio, con un gran potencial de beneficiar a personas de bajos recursos.
* **Competencia:** Alta, ya que existen diversos sistemas de recetas en el mercado.
* **Tecnología:** Disponible y accesible.
* **Recursos internos:** Dependerá de la capacidad técnica de la organización.
* **Riesgo:** Alto, debido a la complejidad del proyecto y la necesidad de financiación.

**Beneficios:**

* Mayor acceso a medicamentos para personas de bajos recursos.
* Reducción del tiempo de espera para obtener recetas.
* Disminución de errores en la prescripción de medicamentos.
* Mejora en la salud de la población objetivo.

**Oportunidad 3**

**Realizar un sistema de control de elementos ortopédicos para una organización que tiene como objetivo combatir la parálisis infantil.**

**Análisis:**

* **Mercado:** Específico, pero con un alto impacto en la calidad de vida de las personas que necesitan elementos ortopédicos.
* **Competencia:** Baja, ya que este tipo de software suele ser personalizado.
* **Tecnología:** Disponible y accesible.
* **Recursos internos:** Dependerá de la capacidad técnica de la organización.
* **Riesgo:** Medio.

**Beneficios:**

* Mejor control del inventario de elementos ortopédicos.
* Mayor eficiencia en la entrega de elementos ortopédicos.
* Reducción de errores en la gestión de los elementos ortopédicos.
* Mejora en la calidad de vida de las personas que necesitan este tipo de ayuda.

**Recomendaciones:**

* **Analizar en detalle las necesidades de cada una de las organizaciones.**
* **Considerar el presupuesto disponible para cada proyecto.**
* **Evaluar la capacidad técnica interna para desarrollar e implementar el software.**
* **Buscar la colaboración de empresas o entidades que puedan aportar recursos o financiación.**

**Conclusión**

Las tres oportunidades de desarrollo de software que se han presentado tienen un gran potencial para mejorar la vida de las personas. La decisión final sobre qué proyecto desarrollar dependerá de una evaluación cuidadosa de los factores mencionados anteriormente.

Selección de desarrollo

Se seleccionará la oportunidad n° 3 mencionada anteriormente en el módulo anterior para su comienzo de planificación que posteriormente será desarrollado.

Documento de Problemática y Solución con Software

Problemática

La organización enfrenta desafíos significativos en la gestión eficiente de elementos y recursos, como sillas de ruedas y otros equipos ortopédicos.

A pesar de tener personal comprometido, carecen de herramientas informáticas para llevar un control preciso de inventario.

En lo que se necesitaría es:

* Controlar la cantidad de los elementos entregados, devueltos y en depósito.
* Generar comprobantes automáticos
* Recibir notificaciones sobre el estado y vencimiento de los elementos.

Alcance de la Solución:

La solución propuesta es un software de gestión integral que abordará la problemática desde la entrada de datos hasta la generación de comprobantes y notificaciones. El sistema se enfocará en la optimización de procesos para garantizar un uso eficiente de los recursos disponibles.

**Procesos Detallados**

1. Registro de Elementos

Descripción: Ingreso de datos sobre sillas de ruedas y otros elementos.

Objetivo: Mantener un inventario actualizado y preciso.

Detalles:

Captura de información:

* Tipo de elemento (silla de ruedas, muletas, andadores, etc.).
* Nombre y apellido del beneficiario.
* Fecha de adquisición.
* Número de teléfono.
* Ubicación actual del elemento (almacén, en uso).

Flujo del proceso:

1. Recepción del elemento.
2. Ingreso de datos en el sistema.
3. Verificación y validación de la información.
4. Actualización del inventario.

2. Generación de Comprobantes

Descripción: Creación automática de comprobantes al retirar o devolver un elemento.

Objetivo: Proporcionar un registro tangible y verificable de las transacciones.

Detalles:

Datos incluidos en el comprobante:

* Información del beneficiario (nombre, apellido, identificación).
* Detalles del elemento (tipo, estado).
* Fecha y hora de la transacción.
* Firmas digitales o físicas (si se requiere).

Flujo del proceso:

1. Solicitud de retiro/devolución del elemento.
2. Generación automática del comprobante.
3. Verificación de los datos.
4. Impresión y entrega del comprobante.
5. Registro digital del comprobante en el sistema.

3. Notificaciones de Vencimiento

Descripción: Sistema de alerta para elementos próximos a vencer.

Objetivo: Evitar el uso de elementos caducados y planificar renovaciones.

Detalles:

Tipos de notificaciones:

* Vencimiento de préstamos de elementos.
* Renovación de registros de beneficiarios.

Frecuencia de las notificaciones:

* Mensuales, semanales, o diarias según la proximidad del vencimiento.

Canales de notificación:

* Alertas en el sistema.

Flujo del proceso:

1. Monitoreo continuo del estado y fechas de vencimiento.
2. Generación de alertas según los criterios establecidos.
3. Envío de notificaciones a los usuarios correspondientes.
4. Seguimiento y confirmación de acciones realizadas.

4. Consulta de Inventario

Descripción: Acceso rápido y fácil a la información del inventario.

Objetivo: Facilitar la toma de decisiones y la planificación.

Detalles:

Funciones de búsqueda y filtro:

* Búsqueda por tipo de elemento, estado, ubicación, beneficiario, etc.
* Filtros avanzados para obtener vistas detalladas y específicas del inventario.

Reportes y análisis:

* Generación de reportes periódicos sobre el estado del inventario.
* Análisis de tendencias y uso de los elementos.

Flujo del proceso:

1. Ingreso al módulo de consulta del inventario.
2. Aplicación de criterios de búsqueda y filtros.
3. Visualización de resultados.
4. Generación de reportes y análisis, si es necesario.

5. Proceso de Beneficiarios

Descripción: Registro y seguimiento de los beneficiarios de los elementos.

Objetivo: Simplificar la administración y gestión de los beneficiarios.

Detalles:

Información del beneficiario:

* Datos personales (nombre, identificación, contacto).
* Historial de préstamos y devoluciones.
* Requisitos específicos y notas médicas.

Módulo de gestión:

* + Inscripción y actualización de datos.
  + Renovación de registros y seguimiento de necesidades.
  + Comunicación con los beneficiarios.

Flujo del proceso:

1. Registro del beneficiario en el sistema.
2. Actualización continúa de la información.
3. Seguimiento de préstamos y devoluciones.
4. Renovación de registros según las políticas de la organización.

Conclusión

La implementación de este sistema de gestión de inventario permitirá a la organización gestionar eficazmente sus recursos, mejorando la visibilidad y control sobre el inventario. Además, automatizará procesos críticos como la generación de comprobantes y las notificaciones de vencimiento, facilitando la planificación y el cumplimiento de trámites administrativos. Esto contribuirá a un funcionamiento más eficiente y organizado, optimizando el uso de recursos y mejorando la atención a los beneficiarios.

**Módulo II**

Proceso de Desarrollo del Sistema

1. Delegación de Responsabilidades

A. Frontend (Interfaz de Usuario)

Responsable

* Román Valle

Descripción

* Desarrollo de la interfaz de usuario para el sistema de gestión de inventario.

Tareas Asignadas

Diseño y Maquetado de Páginas

* Crear wireframes y prototipos de las interfaces.
* Diseñar una interfaz de usuario intuitiva y fácil de usar.
* Asegurar la consistencia visual y la usabilidad de las páginas.

Implementación de la Lógica de Presentación

* Programar las vistas y componentes utilizando tecnologías como HTML, CSS y JavaScript.
* Integrar frameworks y bibliotecas de frontend (React, Vue.js, Angular).
* Gestionar el estado de la aplicación y la interacción del usuario.

Integración con el Backend

* Implementar llamadas a la API para interactuar con el backend.
* Manejar la autenticación y autorización del usuario.
* Asegurar la correcta visualización y actualización de datos provenientes del backend.

B. Backend

Responsable

* Tobías Lazos

Descripción

* Desarrollo del backend
* Desarrollo de la API utilizando Python y el framework Django.

Tareas Asignadas

Implementación de Modelos de Datos

* Diseñar el esquema de la base de datos.
* Crear modelos para representar los elementos, usuarios, transacciones, notificaciones, etc.
* Configurar relaciones y validaciones de datos.

Lógica de Procesamiento de Datos

* Desarrollar la lógica de negocio y las reglas de procesamiento de datos.
* Implementar funciones para gestionar el inventario, transacciones y notificaciones.
* Asegurar la integridad y consistencia de los datos.

Creación de Endpoints de API

* Desarrollar endpoints RESTful para permitir la comunicación con el frontend.
* Asegurar la seguridad de la API mediante autenticación y autorización.
* Implementar documentación de la API utilizando herramientas como Swagger o Django REST Framework.

C. Análisis

Responsables

* Román Valle
* Mateo Molina

Descripción

Evaluación y análisis de requisitos, diseño de la arquitectura del sistema, elaboración de la documentación técnica.

Tareas Asignadas

Reuniones de Planificación y Revisión

* Organizar y participar en reuniones de análisis de requisitos con stakeholders.
* Identificar y documentar los requisitos funcionales y no funcionales.
* Realizar revisiones periódicas del progreso del proyecto y ajustes necesarios.

Diseño de la Arquitectura del Sistema

* Elaborar diagramas de arquitectura del sistema (por ejemplo, diagramas de flujo de datos, diagramas de componentes).
* Definir la estructura y las tecnologías a utilizar en el frontend y backend.
* Planificar la integración entre los diferentes módulos del sistema.

Elaboración de la Documentación Técnica

* Documentar el diseño del sistema y la arquitectura.
* Crear manuales de usuario y guías de desarrollo.
* Mantener la documentación actualizada durante todo el ciclo de vida del proyecto.

2. Proceso de Implementación

A. Configuración del Entorno de Desarrollo

* Configurar entornos de desarrollo, prueba y producción.
* Gestionar repositorios de código fuente y sistemas de control de versiones (por ejemplo, Git).

B. Desarrollo Iterativo

* Utilizar metodologías ágiles (Scrum, Kanban) para gestionar el desarrollo.
* Realizar sprints de desarrollo con entregas incrementales.
* Probar y depurar el código en cada iteración.

C. Pruebas y Validación

* Realizar pruebas unitarias, de integración y de sistema.
* Asegurar la calidad del software mediante pruebas automatizadas y manuales.
* Corregir errores y mejorar el rendimiento del sistema.

D. Despliegue y Mantenimiento

* Desplegar el sistema en el entorno de producción.
* Proporcionar soporte y mantenimiento continuo.
* Planificar actualizaciones y mejoras futuras.

Conclusión

La delegación clara y detallada de responsabilidades asegura una ejecución efectiva del proyecto, con roles bien definidos para cada miembro del equipo. Esto no solo facilita la colaboración y la comunicación, sino que también permite un seguimiento más preciso del progreso y una respuesta rápida a cualquier desafío que surja durante el desarrollo del sistema.

2. Diseño en Figma

Descripción

Creación del prototipo en Figma para diseñar la interfaz de usuario del sistema de gestión de inventario. Este diseño se centra en la experiencia del usuario y en la estructuración lógica de la interfaz, asegurando una navegación intuitiva y eficiente.

Componentes del Diseño

A. Logueo de Usuario

Descripción

Pantalla de inicio de sesión donde los usuarios podrán ingresar sus credenciales para acceder al sistema.

Detalles

* Campos de entrada para nombre de usuario y contraseña.
* Botón de inicio de sesión.

B. Página Principal

Descripción

* Pantalla inicial que muestra un resumen de las notificaciones y el estado general del sistema.

Detalles

* Panel de notificaciones recientes.
* Acceso rápido a las funciones principales.
* Indicadores de estado del inventario y alertas importantes.

C. Sistema de Notificaciones

Descripción

* Módulo para visualizar las notificaciones del sistema.

Detalles

* Lista de notificaciones con detalles como fecha, tipo y descripción.

D. Páginas de Préstamos, Beneficiarios e Implementos

Descripción

* Pantallas dedicadas a la gestión de préstamos, beneficiarios y el inventario de implementos.

Detalles

Préstamos

* Lista de elementos prestados con detalles del beneficiario y fecha de préstamo.
* Botones para registrar nuevas entregas y devoluciones.
* Opciones para filtrar y buscar préstamos específicos.

Beneficiarios

* Lista de beneficiarios con información personal.
* Formularios para agregar, editar o eliminar beneficiarios.
* Filtros y búsqueda avanzada para gestionar beneficiarios.

Implementos

* + Vista del inventario con detalles de cada elemento.
  + Funcionalidades para agregar, editar o dar de baja implementos.
  + Herramientas para buscar y filtrar implementos según criterios específicos.

E. Sistema de Historial

Descripción

* Módulo que muestra el historial de todas las transacciones y actividades del sistema.

Detalles

* Registro cronológico de entregas, devoluciones y mantenimientos.
* Detalles de cada transacción, incluyendo fechas y acciones realizadas.

F. Sistema para Altas, Bajas y Modificaciones

Descripción

* Módulo que permite la gestión completa de los elementos del inventario, incluyendo altas, bajas y modificaciones.

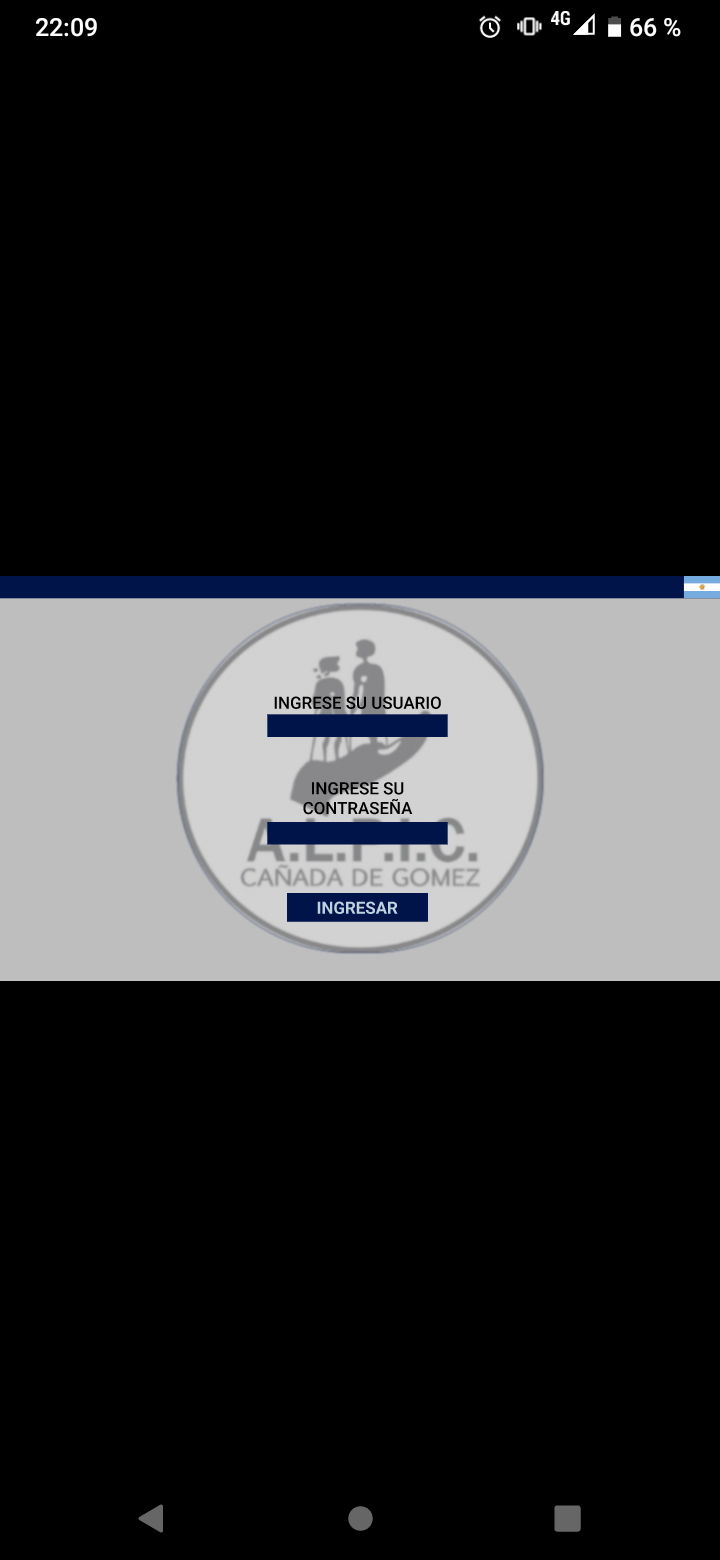
Detalles

* Formularios para registrar nuevos elementos en el inventario.
* Opciones para dar de baja elementos obsoletos o dañados.
* Funcionalidades para modificar la información de los elementos existentes.

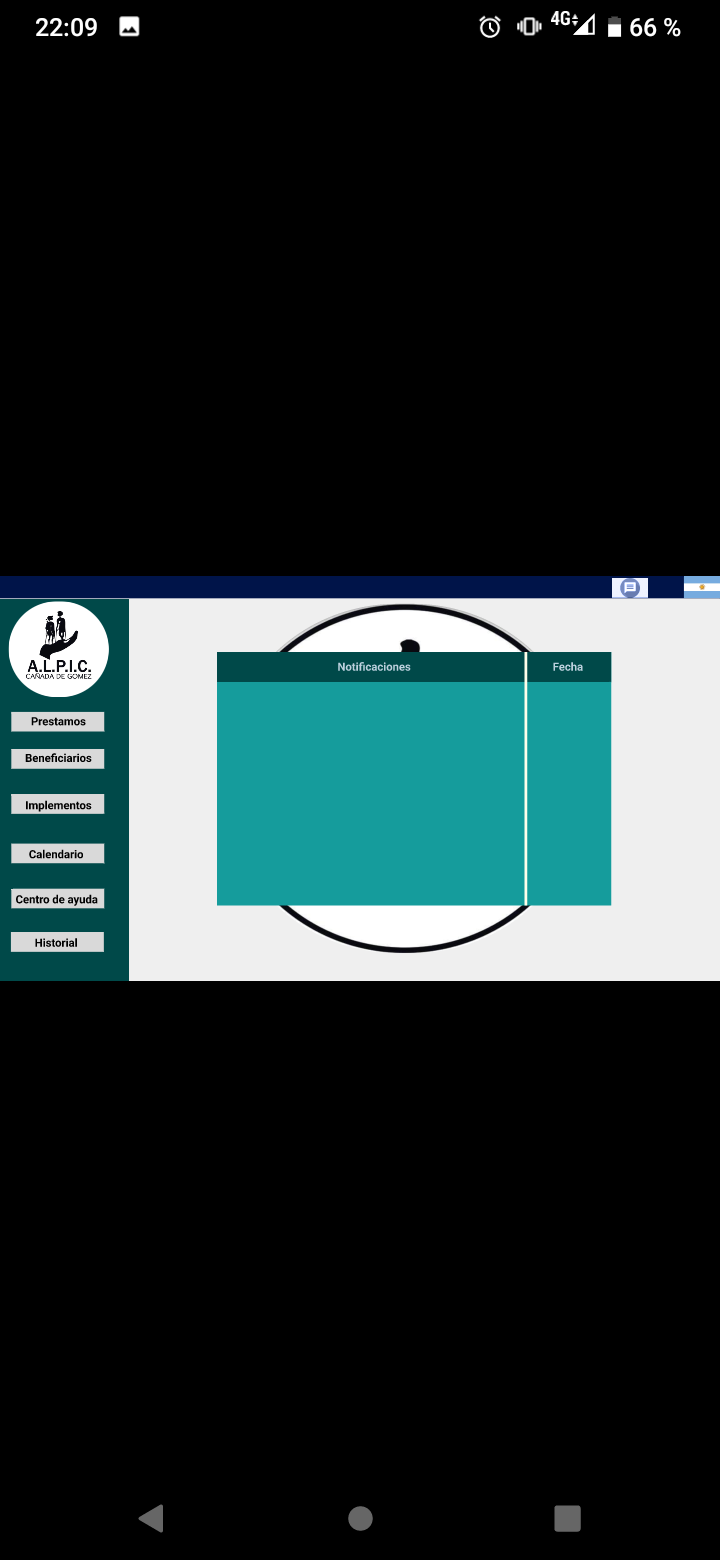
Conclusión

El diseño en Figma proporciona una representación visual detallada de la interfaz de usuario del sistema de gestión de inventario. Cada componente está diseñado para maximizar la usabilidad y eficiencia, asegurando que los usuarios puedan navegar y utilizar el sistema de manera intuitiva. A continuación se presentan los prototipos visuales de cada sección:

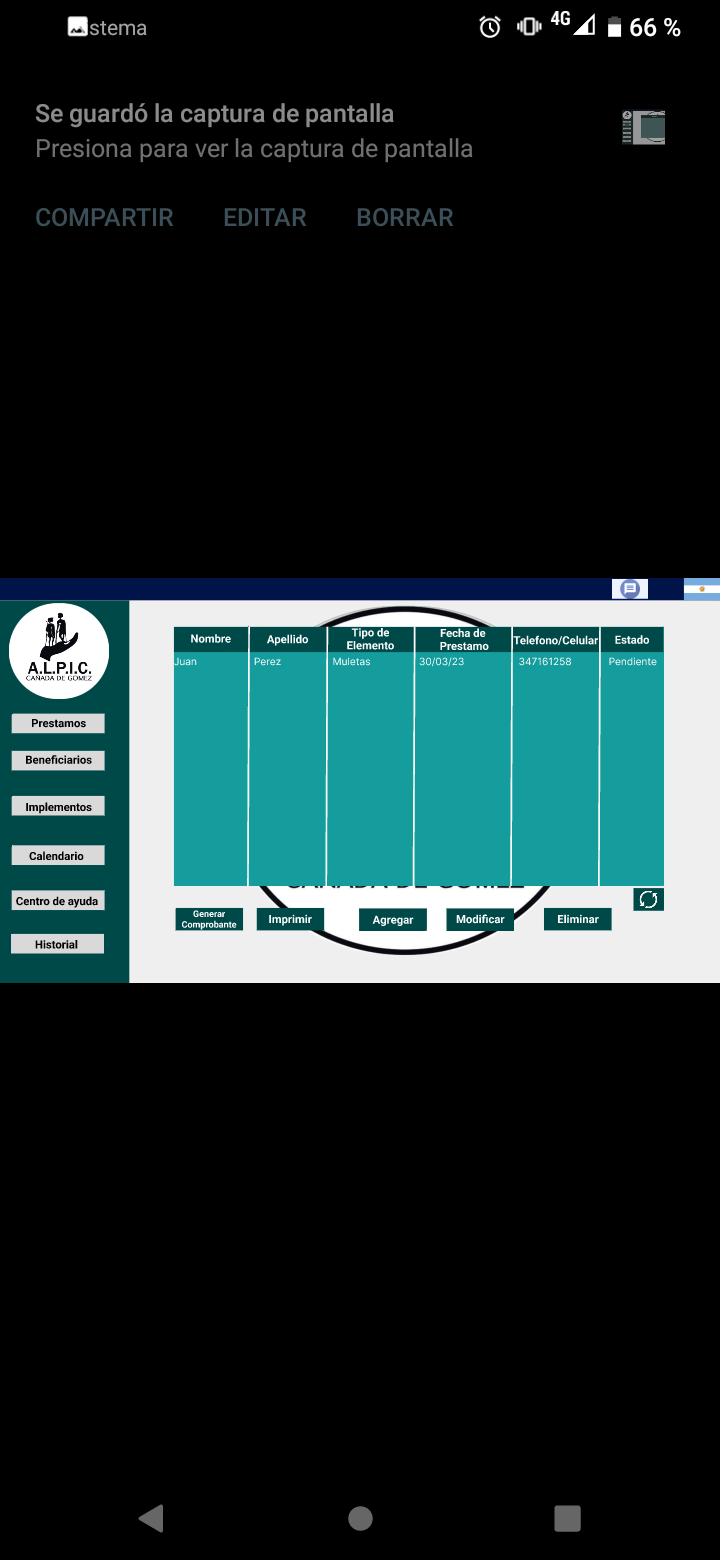
Logueo



Interfaz principal



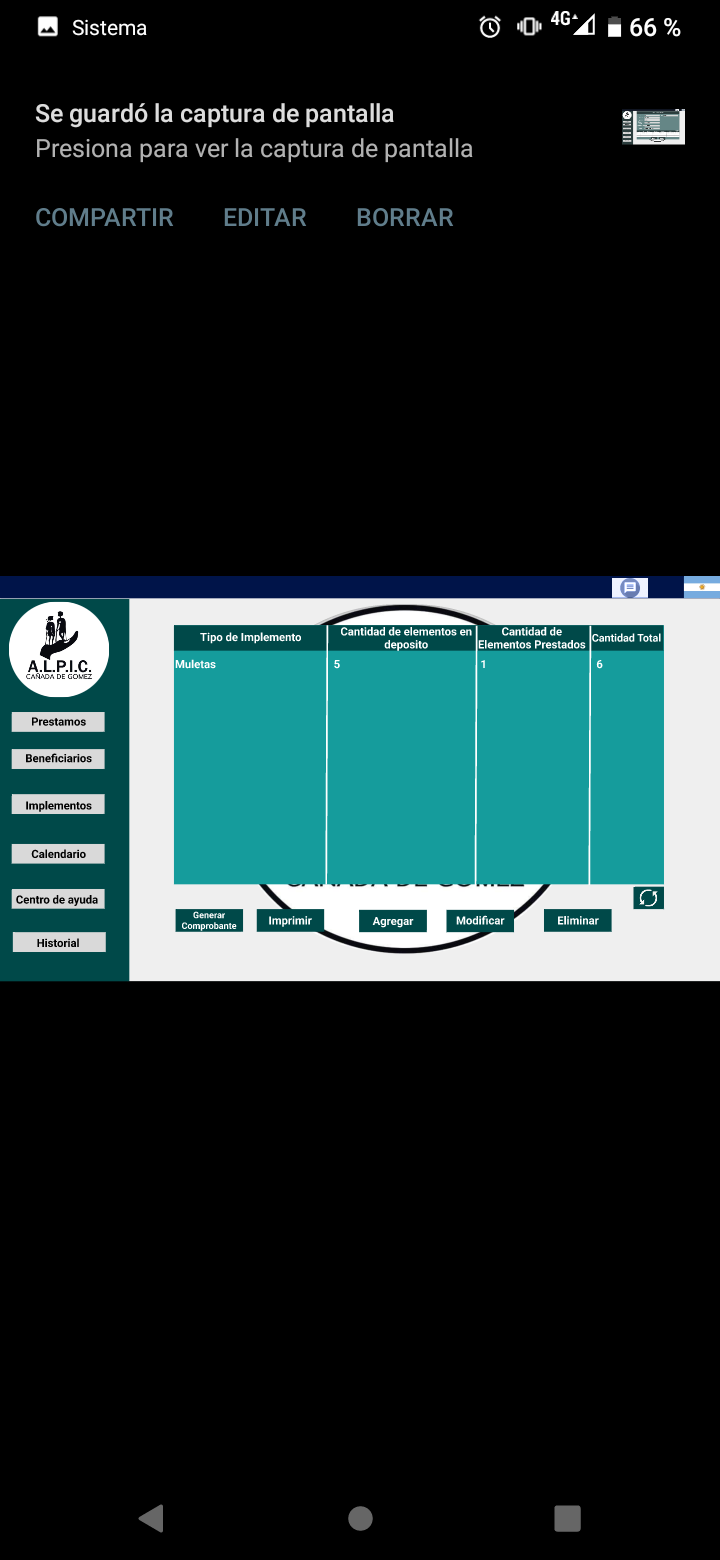
Prestamos



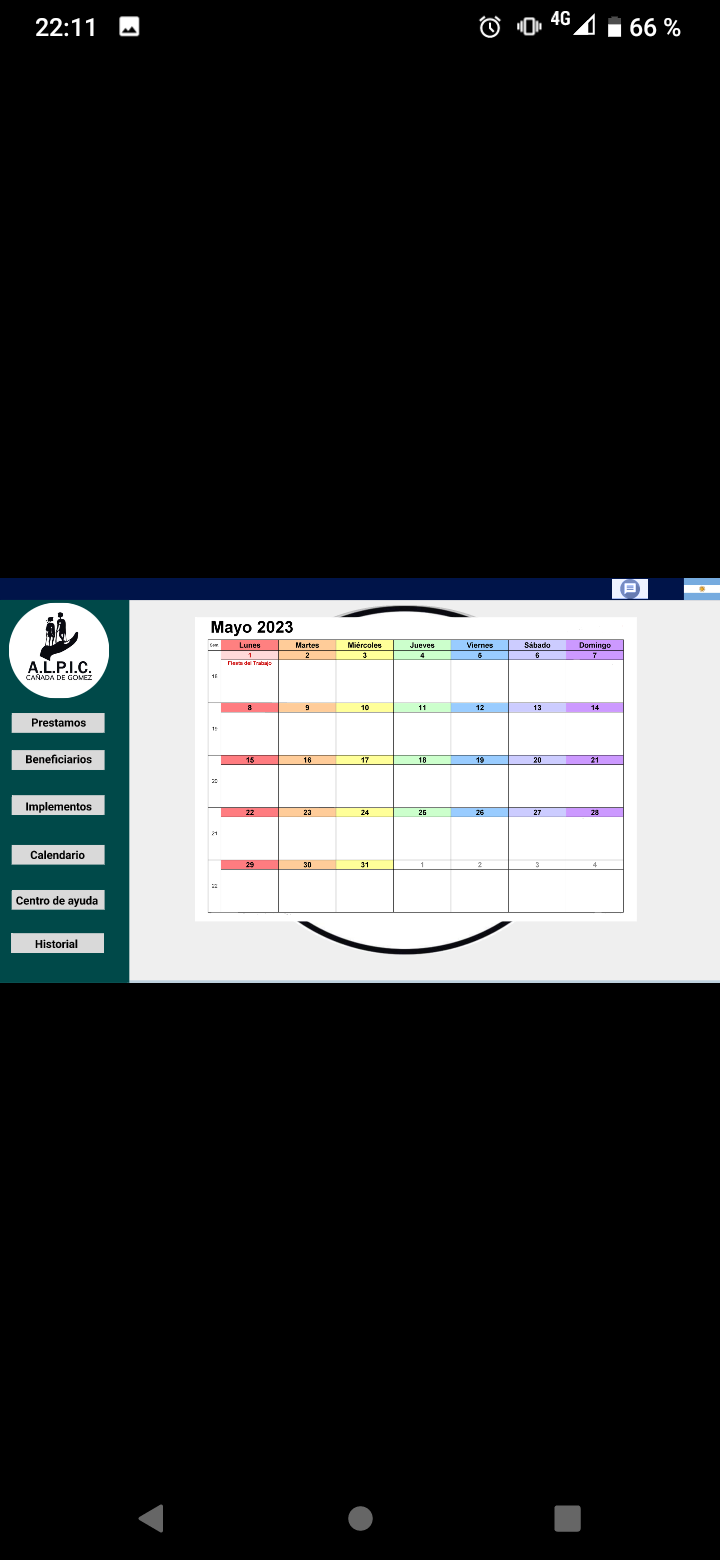
Beneficiarios



Implementos



Calendario



Ayuda



Notificaciones



Agregar préstamos



Modificar Préstamos



Eliminar Préstamos



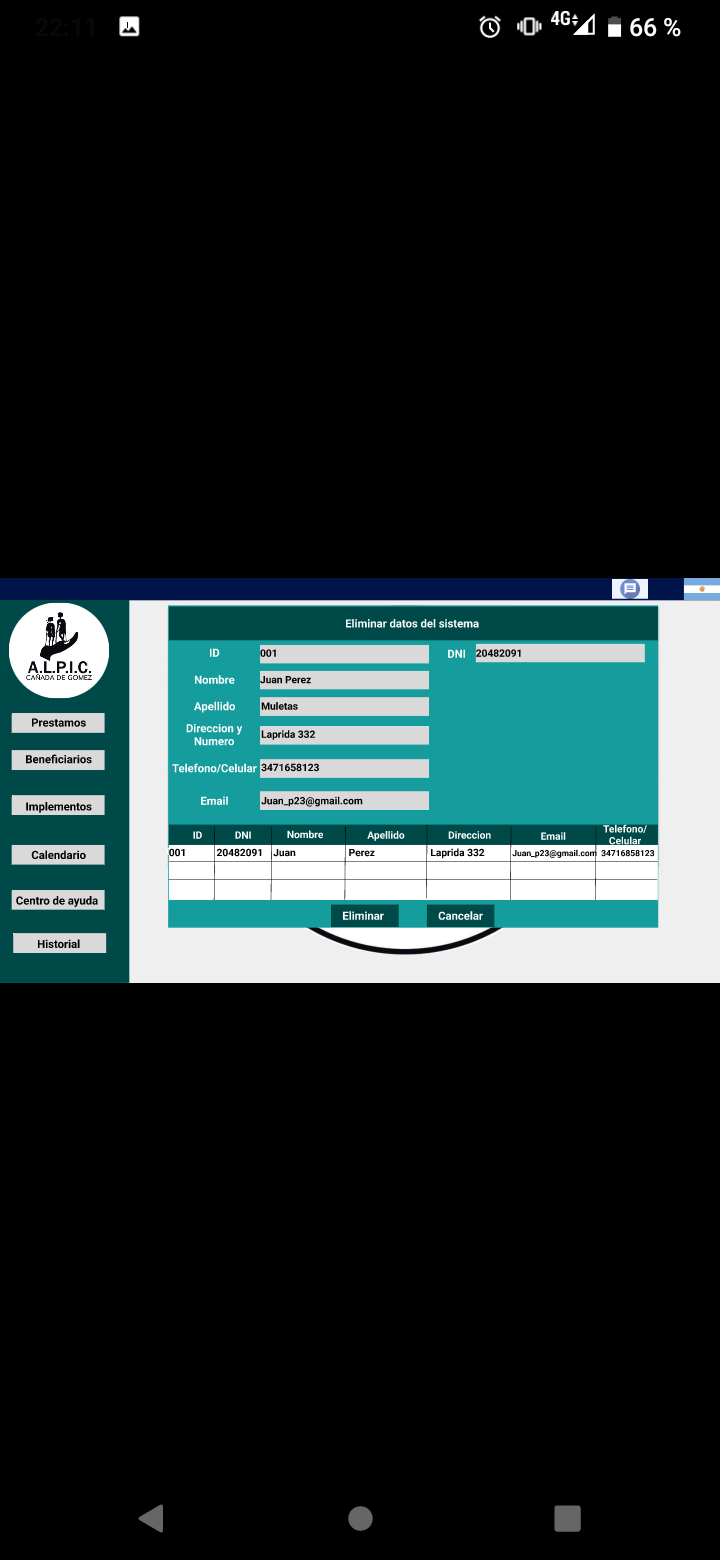
Agregar beneficiarios



Modificar beneficiarios



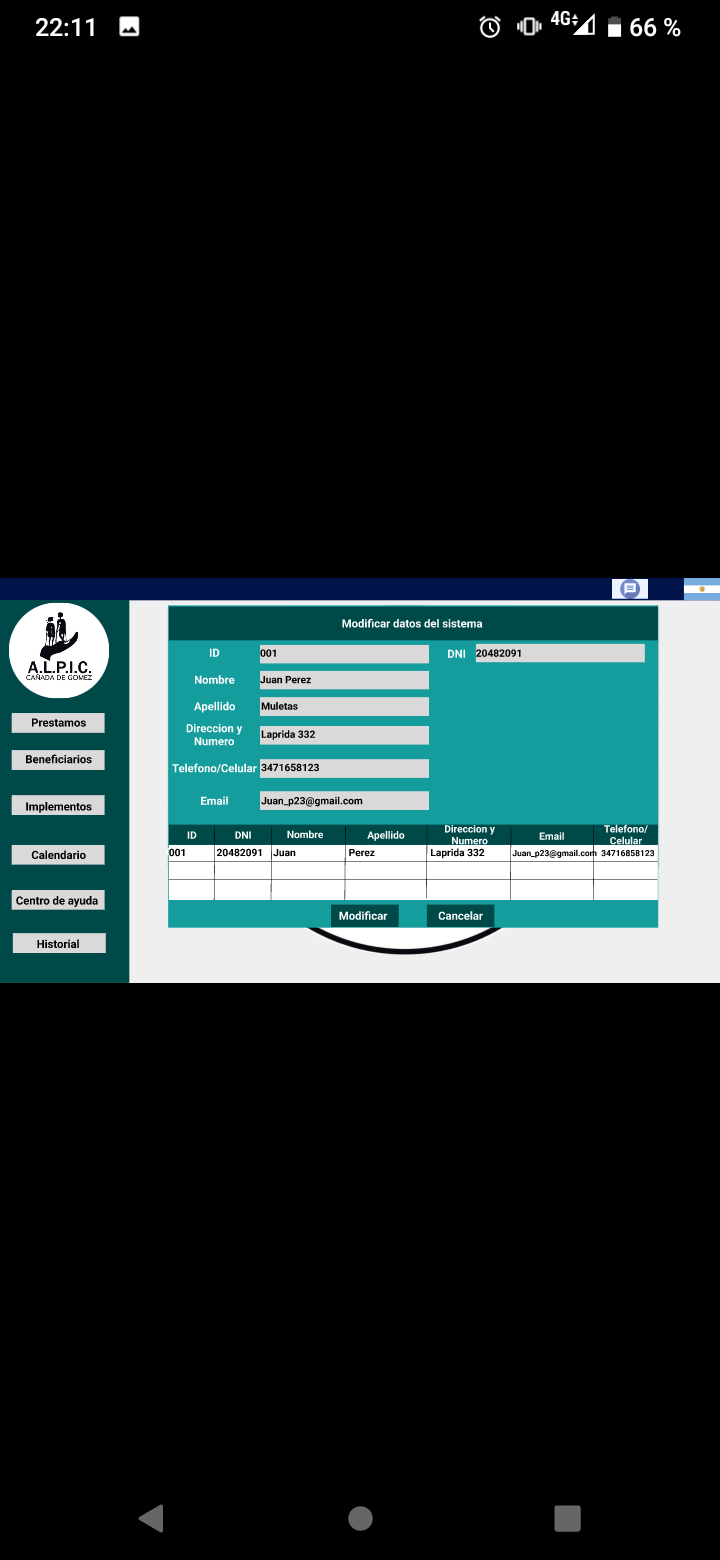
Eliminar beneficiarios



Agregar implementos



Modificar implementos



Eliminar implementos



3. Reunión de Revisión de Diseño:

Fecha y hora

* 19:00 hs del 07 de Junio de 2023.

Asistentes

* Román Valle (Desarrollador Frontend)
* Tobías Lazos (Desarrollador Backend)
* Mateo Molina (Analista)
* Integrantes de la organización
* Profesor a cargo

Objetivo

* Obtener feedback y validación sobre el diseño del sistema.

Agenda

1. Presentación del diseño
2. Discusión sobre posibles ajustes
3. Confirmación de los siguientes pasos

Resumen de la Reunión

**1.** Presentación del Diseño del Sistema

* Descripción del proceso:
  + Se mostró un recorrido completo del prototipo en Figma, incluyendo las diferentes pantallas y funcionalidades del sistema.
  + Cada componente del diseño (logueo, página principal, sistema de notificaciones, gestión de préstamos, beneficiarios, implementos, historial y ABM) fue explicado en detalle, destacando su funcionalidad y la experiencia de usuario esperada.

**2.** Feedback Recibido

* Comentarios generales:
  + La organización brindó un feedback positivo respecto a la estructura y usabilidad del diseño.
  + Se destacaron aspectos como la claridad de la interfaz, la facilidad de navegación y la lógica en la disposición de los elementos.
* Puntos específicos discutidos:
  + Disposición de elementos: No se sugirieron cambios significativos en la disposición de elementos en las pantallas.
  + Paleta de colores: No se propusieron ajustes en los colores utilizados para los componentes.

3. Confirmación de los Siguientes Pasos

Comunicación continúa:

* + Se acordó mantener una comunicación abierta durante todo el desarrollo del sistema.
  + Se estableció la necesidad de realizar reuniones periódicas para mostrar avances, discutir problemas y recibir feedback continuo.

Acciones Siguientes

1. Establecimiento de Calendario de Reuniones Regulares:

* Frecuencia: Quincenal.
* Propósito: Presentación de avances, discusión de problemas, y ajuste del plan de desarrollo según feedback recibido.

1. Continuación del Desarrollo del Sistema:

* Seguir con el desarrollo del frontend y backend conforme a lo planificado.
* Implementar las funcionalidades de acuerdo con los diseños presentados y validados.
* Realizar pruebas continuas para asegurar la calidad y funcionalidad del sistema.

Nota Final

La reunión fue considerada un éxito ya que se obtuvo la validación necesaria para proceder con el desarrollo del proyecto. Todos los participantes expresaron su satisfacción con el diseño presentado y se mostraron entusiasmados por la versión final del sistema. Se resaltó la importancia de mantener la comunicación y la colaboración continua para asegurar el éxito del proyecto.

4. Convertir a Código (Django y Python):

Descripción General

La implementación del diseño en código se realizará utilizando el framework Django con el lenguaje de programación Python. Se utilizará HTML para la creación de la interfaz de usuario, junto con CSS y JavaScript para mejorar la experiencia del usuario. El desarrollo incluirá la lógica de la aplicación y su integración con la base de datos.

1. Configuración del Proyecto Django

a.Creación del Proyecto:

* Iniciar un nuevo proyecto Django utilizando el comando django-admin startproject.
* Configurar el entorno virtual para gestionar las dependencias del proyecto.

b. Configuración de la Base de Datos:

* Selección del motor de base de datos (por ejemplo, PostgreSQL, MySQL, SQLite).
* Configuración de las credenciales y parámetros de la base de datos en el archivo settings.py.
* Creación y ejecución de migraciones iniciales para establecer las tablas de la base de datos.

2. Desarrollo del Backend

a.Modelos de Datos:

* Definición de modelos en models.py para representar los elementos, usuarios, transacciones, notificaciones, etc.
* Establecimiento de relaciones entre modelos utilizando claves foráneas y muchos a muchos.
* Implementación de validaciones y métodos personalizados dentro de los modelos.

b. Lógica de Negocio y Procesamiento de Datos:

* Implementación de la lógica de negocio en los modelos y vistas.
* Desarrollo de funciones y métodos para gestionar el inventario, préstamos, devoluciones, y notificaciones.
* Utilización de señales de Django para manejar eventos específicos (por ejemplo, enviar notificaciones al vencer un préstamo).

c. Creación de Endpoints de API:

* Utilización de Django REST Framework para crear endpoints RESTful.
* Implementación de serializadores para convertir modelos a formatos JSON y viceversa.
* Configuración de permisos y autenticación para asegurar la API.

3. Desarrollo del Frontend

a. Diseño de Plantillas HTML:

* Creación de plantillas HTML para las diferentes vistas (login, página principal, préstamos, beneficiarios, implementos, historial, calendario, agregar prestamos, modificar prestamos, eliminar prestamos, agregar beneficiarios, modificar beneficiarios, eliminar beneficiarios, agregar implementos, modificar implementos, eliminar implementos, centro de ayuda).
* Utilización del sistema de plantillas de Django para renderizar datos dinámicos en las vistas.
* Inclusión de archivos estáticos (CSS y JavaScript) para mejorar la interfaz de usuario.

b.Integración con el Backend:

* Implementación de llamadas AJAX para interactuar con la API sin recargar la página.
* Manejo de sesiones y tokens de autenticación para mantener la seguridad del usuario.
* Actualización dinámica del contenido en función de las respuestas de la API.

4. Gestión de Usuarios y Autenticación

a. Sistema de Autenticación:

* Configuración de Django para manejar la autenticación de usuarios.
* Implementación de vistas y formularios de login.
* Utilización de middleware para asegurar que solo los usuarios autenticados puedan acceder a ciertas vistas.

b. Control de Acceso y Permisos:

* Definición de grupos y permisos para diferentes tipos de usuarios (administradores, usuarios normales).
* Implementación de decoradores y clases basadas en vistas para controlar el acceso a las funcionalidades del sistema.

5.Notificaciones y Alertas

a. Sistema de Notificaciones:

* Implementación de un sistema para generar y almacenar notificaciones en la base de datos.
* Creación de vistas y plantillas para mostrar notificaciones al usuario.
* Configuración de tareas periódicas utilizando Celery para enviar alertas de vencimiento.

6. Pruebas y Validación

a. Pruebas Unitarias:

* Escribir pruebas unitarias para modelos, vistas y serializadores.
* Utilización de la biblioteca unittest de Python y pytest para ejecutar las pruebas.

b. Pruebas de Integración:

* Realización de pruebas de integración para asegurar que todos los componentes del sistema funcionen juntos correctamente.
* Utilización de herramientas como Selenium para pruebas de interfaz de usuario.

c. Pruebas de Usuario y Retroalimentación:

* Despliegue del sistema en un entorno de prueba y recolección de feedback de los usuarios.
* Ajuste y mejora del sistema basado en la retroalimentación recibida.

7. Despliegue y Mantenimiento

a. Preparación para el Despliegue:

* Configuración de archivos settings.py para producción (configuración de seguridad, compresión de archivos estáticos).
* Uso de herramientas como Docker para crear contenedores del sistema.

b. Despliegue en el Servidor:

* Selección de un proveedor de servicios de alojamiento.
* Configuración del servidor y despliegue del sistema.

c. Mantenimiento Continuo:

* Monitorización del sistema para detectar y corregir errores.
* Planificación de actualizaciones y mejoras continúas.
* Realización de copias de seguridad periódicas de la base de datos.

5. Desarrollo Simultáneo de Diseño y Base de Datos:

Descripción General

El desarrollo simultáneo del diseño de la interfaz de usuario y la estructura de la base de datos es un enfoque integral que asegura la coherencia y funcionalidad del sistema. Este enfoque permite que ambas partes del desarrollo se informen y ajusten mutuamente, garantizando una integración perfecta entre la interfaz de usuario y el backend.

1. Planificación y Coordinación

a.Definición de Requisitos:

* Requisitos Funcionales: Documentación de todas las funcionalidades necesarias, como el registro y seguimiento de elementos, la gestión de beneficiarios, la generación de notificaciones y comprobantes.
* Requisitos No Funcionales: Establecimiento de criterios de rendimiento, seguridad y usabilidad.

b. Especificaciones Técnicas:

* Diseño de la Base de Datos: Definición de esquemas de base de datos, relaciones entre tablas, índices y restricciones.
* Diseño de la Interfaz de Usuario: Creación de wireframes y prototipos interactivos que definan la estructura y flujo de las pantallas.

c. Equipo y Responsabilidades:

* Encargado del Backend: Desarrollo y mantenimiento de la base de datos y la lógica del servidor.
* Encargado del Frontend: Diseño y desarrollo de la interfaz de usuario, asegurando una experiencia de usuario coherente y atractiva.
* Coordinación: Reuniones regulares entre los encargados de frontend y backend para garantizar la coherencia y sincronización del desarrollo.

2. Desarrollo de la Interfaz de Usuario

a.Herramientas y Tecnologías:

* HTML, CSS y JavaScript: Para la estructura, estilos y comportamiento interactivo de las páginas.
* Figma: Para el diseño y prototipado de la interfaz de usuario.

b. Implementación de Páginas y Componentes:

* Login de Usuario: Formulario de autenticación con validación de campos y gestión de sesiones.
* Página Principal: Panel de control con notificaciones, indicadores y accesos rápidos.
* Sistema de Notificaciones: Visualización y notificaciones del sistema.
* Gestión de Préstamos, Beneficiarios e Implementos: Formularios y tablas para el manejo de datos de préstamos, beneficiarios e implementos.
* Sistema de Historial: Registro de transacciones y actividades del sistema.
* Sistema para Altas, Bajas y Modificaciones: Formularios para la gestión del inventario.

c. Pruebas de Interfaz:

* Pruebas de Usabilidad: Evaluación de la facilidad de uso y navegabilidad de la interfaz.
* Pruebas de Compatibilidad: Asegurar que la interfaz funcione correctamente en diferentes navegadores y dispositivos.

3. Desarrollo de la Base de Datos

a. Herramientas y Tecnologías:

* Sistema de Gestión de Bases de Datos (SGBD): PostgreSQL, MySQL, o SQLite para la gestión y almacenamiento de datos.
* Django ORM: Para la definición y manipulación de los modelos de datos en Django.

b. Diseño de Esquema de Base de Datos:

* Modelado de Datos: Creación de diagramas entidad-relación (ER) para definir las tablas y sus relaciones.
* Normalización: Asegurarse de que los datos estén organizados eficientemente para evitar redundancias y asegurar la integridad.
* Índices y Restricciones: Definición de índices para mejorar el rendimiento y restricciones para asegurar la integridad de los datos.

c. Implementación de Modelos:

* Definición de Modelos: Creación de modelos en models.py en Django para representar las tablas de la base de datos.
* Migraciones: Uso de makemigrations y migrate para aplicar cambios en la estructura de la base de datos.
* Relaciones: Establecimiento de relaciones entre modelos utilizando claves foráneas y relaciones muchos a muchos.

d. Pruebas de Base de Datos:

* Pruebas de Integridad: Asegurar que las restricciones y relaciones se mantengan correctamente.
* Pruebas de Rendimiento: Evaluar la eficiencia de las consultas y el rendimiento general de la base de datos.

4. Integración y Coherencia

a. Sincronización de Frontend y Backend:

* Endpoints de API: Creación de endpoints en Django REST Framework para proporcionar datos al frontend.
* Consumo de API: Implementación de llamadas AJAX o uso de librerías como Axios en el frontend para interactuar con la API.
* Validaciones y Manejo de Errores: Asegurar que los datos ingresados por el usuario sean validados tanto en el frontend como en el backend.

b. Pruebas de Integración:

* Pruebas de Flujo Completo: Simulación de escenarios completos de uso para asegurar que todos los componentes funcionen juntos sin problemas.
* Corrección de Errores: Identificación y resolución de inconsistencias o errores en la comunicación entre frontend y backend.

5. Revisión y Ajustes Continuos

a. Feedback Continuo:

* Reuniones Periódicas: Reuniones regulares para revisar el progreso y ajustar los planes según sea necesario.
* Revisión por Pares: Evaluación del código y diseño por parte de otros desarrolladores para asegurar la calidad y consistencia.

b.Ajustes Basados en Feedback:

* Iteraciones de Diseño: Ajustes y mejoras en el diseño de la interfaz basados en pruebas de usuario y feedback.
* Optimización de Base de Datos: Ajustes en la estructura y consultas de la base de datos para mejorar el rendimiento.

Conclusión

El desarrollo simultáneo del diseño de la interfaz de usuario y la base de datos es crucial para garantizar un sistema cohesivo y funcional. La coordinación continua entre los equipos de frontend y backend, junto con un enfoque iterativo y basado en feedback, asegura que el sistema final cumpla con los requisitos y expectativas de los usuarios. Este enfoque integral permite una integración perfecta entre la interfaz de usuario y la base de datos, garantizando la eficiencia, usabilidad y rendimiento del sistema.

6. Reunión de Avances en la Programación:

Fecha y Hora: 19:00 hs del 11 de Octubre de 2023.

Asistentes:

* Román Valle (Desarrollador Frontend)
* Tobías Lazos (Desarrollador Backend)
* Mateo Molina (Analista)
* Integrantes de la organización
* Profesor a cargo

Objetivo

Presentar los avances en la programación y obtener feedback sobre el progreso del proyecto.

Agenda

1. Demostración de Funcionalidades Implementadas:
   * Presentación detallada de las características y módulos desarrollados hasta el momento.
   * Ejemplos prácticos de cómo los usuarios interactuarán con el sistema.
   * Explicación de la lógica y arquitectura detrás de las implementaciones.
2. Discusión sobre la Satisfacción con los Avances:
   * Evaluación de la alineación del progreso actual con los requisitos y expectativas iniciales.
   * Identificación de áreas que cumplen o exceden las expectativas.
3. Recopilación de Feedback y Sugerencias para Mejoras:
   * Recepción de comentarios específicos sobre las funcionalidades demostradas.
   * Discusión de sugerencias para optimización y mejoras adicionales.
   * Priorización de mejoras basadas en la retroalimentación recibida.

Resumen de la Reunión

1. Presentación de Funcionalidades Implementadas:

* Aspectos Clave Demostrados:
  + Sistema de autenticación e ingreso de usuario.
  + Página principal con panel de control y notificaciones en tiempo real.
  + Módulos de gestión de préstamos, beneficiarios e implementos.
  + Sistema de historial y registro de actividades.
  + Sistema de calendario para visualización de las fechas.
  + Interfaz de usuario intuitiva y coherente con el diseño aprobado.

2. Feedback Recibido:

* Comentarios Positivos:
  + Los miembros de la organización expresaron su satisfacción con la facilidad de uso y funcionalidad de las interfaces.
  + Se destacó la eficiencia y rapidez del sistema en las pruebas realizadas.
* Áreas de Mejora:
  + Se sugirieron optimizaciones en la navegación y disposición de ciertos elementos en la interfaz.
  + Se propuso la incorporación de filtros más avanzados en los módulos de búsqueda.
  + Se mencionó la posibilidad de mejorar el rendimiento en consultas complejas.

3. Discusión de Puntos Específicos:

* Interfaz de Usuario:
  + Ajustes en el diseño visual para mejorar la claridad y accesibilidad.
  + Simplificación de ciertos flujos de usuario para aumentar la eficiencia operativa.
* Optimización de Funcionalidades:
* Mejoras en la lógica de backend para acelerar tiempos de respuesta.
* Ajustes en la base de datos para optimizar consultas y manejo de grandes volúmenes de datos.

4. Acordado para Continuar:

* Mantener el ritmo de desarrollo actual, asegurando la calidad y completitud de las funcionalidades antes de avanzar.
* Establecer una comunicación abierta y continua para abordar problemas y recibir feedback en tiempo real.

Acciones Siguientes

1. Implementación de Mejoras Sugeridas:

* Priorizar y planificar la implementación de las mejoras sugeridas durante la reunión.
* Realizar pruebas de usuario adicionales para validar las mejoras y asegurar la satisfacción del usuario final.

2. Planificación de Reuniones de Seguimiento:

* Establecer un calendario de reuniones para revisar el progreso y ajustar el plan de desarrollo según sea necesario.
* Garantizar que todos los interesados estén actualizados sobre el estado del proyecto y próximos hitos.

3. Continuación del Desarrollo de Nuevas Funcionalidades:

* Seguir trabajando en las funcionalidades pendientes según el plan establecido.
* Asegurar que cada nueva funcionalidad sea probada exhaustivamente antes de ser integrada en el sistema.

Nota Final

La reunión fue considerada un éxito significativo. Aunque se identificaron áreas clave para trabajar, el progreso logrado hasta ahora es considerable y se recibió feedback positivo que asegura un desarrollo prometedor del proyecto. La colaboración continua y el enfoque en la mejora constante serán esenciales para el éxito del proyecto.

Requerimientos de Información

Registro de Usuarios

Datos Requeridos:

* Usuario: String, único, longitud máxima de 30 caracteres. (campo requerido)
* Contraseña: Password, longitud mínima de 8 caracteres, debe incluir mayúsculas, minúsculas, números y caracteres especiales. (campo requerido)

Préstamos

Datos Requeridos:

* Identificador de Préstamo: Integer, clave primaria autoincremental.
* Nombre: String, longitud máxima de 30 caracteres. (campo requerido)
* Apellido: String, longitud máxima de 30 caracteres. (campo requerido)
* Tipo de Elemento: String, longitud máxima de 50 caracteres. (campo requerido)
* Fecha de Préstamo: Date, formato AAAA-MM-DD. (campo requerido)
* Teléfono o Celular: String, longitud máxima de 15 caracteres, validación de formato. (campo requerido)
* Estado: String, posibles valores 'Activo', 'Devuelto', 'Pendiente'. (campo requerido)

Beneficiarios

Datos Requeridos:

* Identificador de Beneficiario: Integer, clave primaria autoincremental.
* Nombre: String, longitud máxima de 30 caracteres. (campo requerido)
* Apellido: String, longitud máxima de 30 caracteres. (campo requerido)
* DNI: String, longitud fija de 8 caracteres, validación de formato. (campo requerido)
* Teléfono o Celular: String, longitud máxima de 15 caracteres, validación de formato. (campo requerido)
* Dirección: String, longitud máxima de 50 caracteres. (campo requerido)
* Estado: String, posibles valores 'Activo', 'Inactivo'. (campo requerido)

Implementos

Datos Requeridos:

* Tipo de Elemento: String, longitud máxima de 50 caracteres. (campo requerido)
* Cantidad de Elementos en Stock: Integer, no negativo. (campo requerido)
* Cantidad de Elementos Entregados: Integer, no negativo. (campo requerido)
* Cantidad Total: Integer, no negativo. (campo requerido)

Historial

Datos Requeridos:

* Registro de Acciones: Text, descripción detallada de la acción realizada. (campo requerido)
* Fecha: Date, formato AAAA-MM-DD. (campo requerido)

Notificaciones

Datos Requeridos:

* Título: String, longitud máxima de 50 caracteres. (campo requerido)
* Fecha: Date, formato AAAA-MM-DD. (campo requerido)

Clasificación de Datos

Tipo de Dato:

* Id: Integer, clave primaria autoincremental.
* Nombre y Apellido: String, longitud máxima de 50 caracteres.
* Dirección: String, longitud máxima de 200 caracteres.
* Usuario: String, único, longitud máxima de 30 caracteres.
* Contraseña: Password, longitud mínima de 8 caracteres, debe incluir mayúsculas, minúsculas, números y caracteres especiales.
* Fecha y Fecha de Préstamo: Date, formato AAAA-MM-DD.
* Tipo de Elemento: String, longitud máxima de 50 caracteres.
* Cantidad de Elementos: Integer, no negativo.
* Cantidad Total: Integer, no negativo.
* Teléfono o Celular: String, longitud máxima de 15 caracteres, validación de formato.
* Estado: String, posibles valores según el contexto ('Activo', 'Inactivo', 'Devuelto', 'Pendiente').

Consideraciones Adicionales:

* Validaciones: Se deben implementar validaciones de formato y restricciones de longitud en el modelo de datos.
* Integridad de Datos: Se deben establecer relaciones y restricciones para asegurar la integridad referencial entre tablas.
* Seguridad: Las contraseñas deben ser almacenadas de forma segura utilizando hashing.
* Indexación: Se deben crear índices en campos frecuentemente consultados para mejorar el rendimiento de las consultas.
* Auditoría: El sistema debe registrar acciones críticas en el historial para permitir auditorías.

Definición de la Base de Datos y Tablas

Tabla de Usuarios

Campos

* id\_usuario
* id\_password

Tabla de Prestamos

Campos

* prestamo.id
* prestamo.nombre
* prestamo.apellido
* prestamo.tipo\_elemento
* prestamo.fecha\_prestamo
* prestamo.telefono\_celular
* prestamo.estado

Tabla de Beneficiarios

Campos

* beneficiarios.id
* beneficiarios.nombre
* beneficiarios.apellido
* beneficiarios.dni
* beneficiarios.telefono
* beneficiarios.direccion
* beneficiarios.estado

Tabla de Implementos

Campos

* implementos.tipo\_elemento
* implementos.cantidad\_elementos
* implementos.cantidad\_elementos\_entregados
* implementos.cantidad\_total

Tabla de Notificaciones

Campos

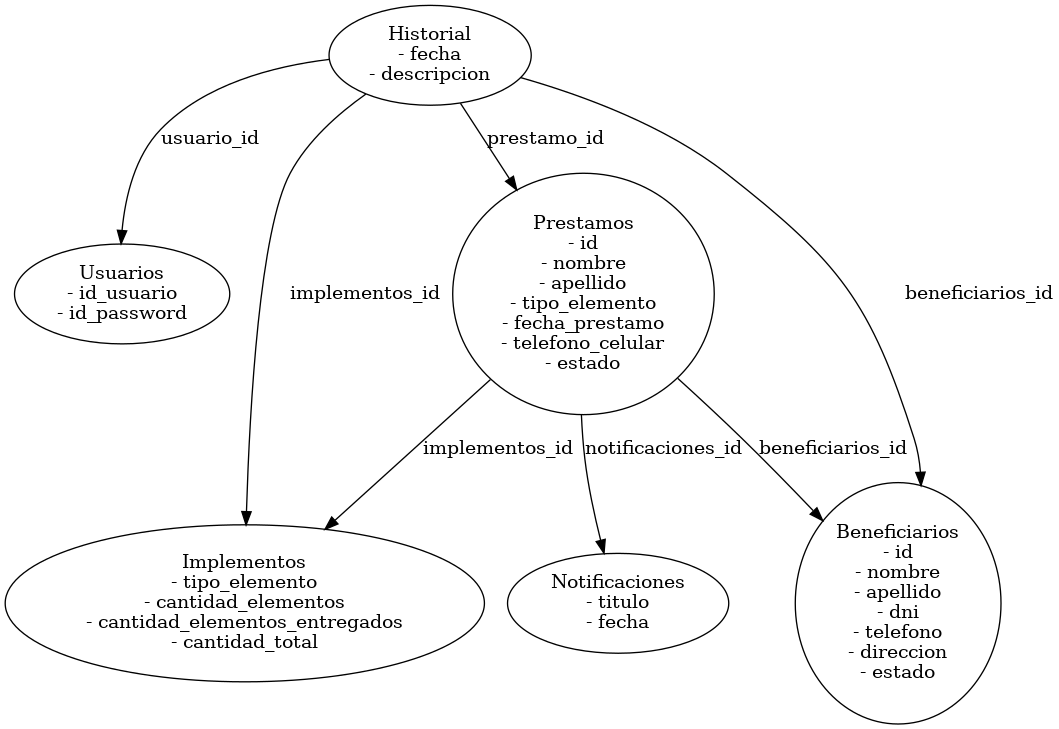
* notificación.titulo
* notificación.fecha

Tabla de Historial

Campos

* registro.fecha
* registro.descripcion

Diagrama de la base de datos



Análisis y defensa de la estructura de la base de datos

Introducción

La base de datos diseñada tiene como objetivo gestionar eficientemente los recursos de una organización, incluyendo elementos ortopédicos como sillas de ruedas, y mantener un control preciso sobre los préstamos, beneficiarios, implementos y notificaciones. La estructura de la base de datos se ha desarrollado para asegurar integridad, eficiencia y escalabilidad.

Estructura de la Base de Datos

1. Tabla de Usuarios
   * Campos:
     + id\_usuario: Integer, clave primaria autoincremental.
     + id\_password: Password, almacenado de forma segura utilizando hashing.
   * Análisis:
     + La tabla de usuarios es fundamental para la autenticación y autorización de los usuarios del sistema. El uso de id\_usuario como clave primaria garantiza unicidad y permite una rápida búsqueda y autenticación. El almacenamiento seguro de contraseñas es crítico para la seguridad del sistema.
2. Tabla de Préstamos
   * Campos:
     + id: Integer, clave primaria autoincremental.
     + nombre: String, longitud máxima de 30 caracteres.
     + apellido: String, longitud máxima de 30 caracteres.
     + tipo\_elemento: String, longitud máxima de 50 caracteres.
     + fecha\_prestamo: Date, formato AAAA-MM-DD.
     + telefono\_celular: String, longitud máxima de 15 caracteres, validación de formato.
     + estado: String, posibles valores 'Activo', 'Devuelto', 'Pendiente'.
     + beneficiarios\_id: Integer, clave foránea, referencia a Beneficiarios.
     + implementos\_id: Integer, clave foránea, referencia a Implementos.
     + notificaciones\_id: Integer, clave foránea, referencia a Notificaciones.
   * Análisis:
     + Esta tabla permite un seguimiento detallado de cada préstamo, incluyendo información sobre el beneficiario y el implemento prestado. Las claves foráneas aseguran la integridad referencial y permiten consultas eficientes para obtener información relacionada.
3. Tabla de Beneficiarios
   * Campos:
     + id: Integer, clave primaria autoincremental.
     + nombre: String, longitud máxima de 30 caracteres.
     + apellido: String, longitud máxima de 30 caracteres.
     + dni: String, longitud fija de 8 caracteres, validación de formato.
     + teléfono: String, longitud máxima de 15 caracteres, validación de formato.
     + dirección: String, longitud máxima de 50 caracteres.
     + estado: String, posibles valores 'Activo', 'Inactivo'.
   * Análisis:
     + La tabla de beneficiarios almacena información esencial sobre las personas que reciben los préstamos. La inclusión de campos detallados como dni y teléfono permite una identificación y comunicación precisas.
4. Tabla de Implementos
   * Campos:
     + tipo\_elemento: String, longitud máxima de 50 caracteres.
     + cantidad\_elementos: Integer, no negativo.
     + cantidad\_elementos\_entregados: Integer, no negativo.
     + cantidad\_total: Integer, no negativo.
   * Análisis:
     + Esta tabla permite un control preciso del inventario de implementos. Los campos cuantitativos permiten un seguimiento detallado del stock y de los implementos entregados, facilitando la gestión de recursos.
5. Tabla de Notificaciones
   * Campos:
     + título: String, longitud máxima de 50 caracteres.
     + fecha: Date, formato AAAA-MM-DD.
   * Análisis:
     + La tabla de notificaciones permite gestionar alertas y recordatorios importantes. El campo titulo describe la notificación, mientras que fecha indica cuándo se debe atender la notificación.
6. Tabla de Historial
   * Campos:
     + fecha: Date, formato AAAA-MM-DD.
     + descripción: Text, descripción detallada de la acción realizada.
     + prestamo\_id: Integer, clave foránea, referencia a Préstamos.
     + beneficiarios\_id: Integer, clave foránea, referencia a Beneficiarios.
     + implementos\_id: Integer, clave foránea, referencia a Implementos.
     + usuario\_id: Integer, clave foránea, referencia a Usuarios.

* Análisis:
  + - Esta tabla registra todas las acciones importantes realizadas en el sistema. Las múltiples claves foráneas aseguran que cada acción esté vinculada a un préstamo, beneficiario, implemento y usuario específicos, proporcionando un registro completo y auditado de las operaciones.

Defensa de la Decisión

1. Integridad Referencial:

* Las claves foráneas en las tablas de Préstamos, Historial, y otras aseguran que los datos estén vinculados correctamente, evitando inconsistencias y asegurando la integridad de la base de datos.

1. Escalabilidad:

* La estructura modular permite una fácil expansión. Nuevas tablas y relaciones pueden añadirse sin afectar significativamente la estructura existente.

1. Eficiencia en Consultas:

* La utilización de claves primarias y foráneas permite la creación de índices que optimizan la velocidad de las consultas, mejorando el rendimiento del sistema.

1. Seguridad:

* El almacenamiento seguro de contraseñas y la auditoría detallada de acciones en el historial mejoran la seguridad y permiten un seguimiento preciso de las actividades.

1. Flexibilidad:

* La base de datos está diseñada para manejar diversas operaciones relacionadas con préstamos, beneficiarios, implementos y notificaciones, permitiendo una gestión completa y eficiente de los recursos.

Conclusión

La estructura de la base de datos propuesta proporciona una solución robusta y eficiente para la gestión de los recursos y operaciones de la organización. La integridad referencial, escalabilidad, eficiencia en consultas, seguridad y flexibilidad son aspectos clave que respaldan la decisión de diseño. Esta base de datos facilitará una gestión precisa y eficiente, mejorando la operación general de la organización.

Diagnóstico de la estructura de la Base de Datos

Introducción

El diagnóstico de la base de datos es una evaluación técnica detallada que busca identificar fortalezas, debilidades, y áreas de mejora. Este análisis incluye una revisión de la estructura de las tablas, las relaciones entre ellas, la integridad de los datos, la seguridad, el rendimiento y la escalabilidad.

Evaluación Técnica

1. Estructura de las Tablas
   * Usuarios
     + Fortalezas:
       - Uso de una clave primaria id\_usuario garantiza unicidad.
       - Almacenamiento seguro de contraseñas.
     + Debilidades:
       - La tabla podría beneficiarse de información adicional, como roles de usuario para una mejor gestión de permisos.
   * Prestamos
     + Fortalezas:
       - Estructura detallada que incluye referencias a beneficiarios, implementos y notificaciones.
       - Campos específicos para seguimiento detallado de préstamos.
     + Debilidades:
       - Podría incluir más detalles, como fecha de devolución esperada o comentarios.
   * Beneficiarios
     + Fortalezas:
       - Detallada información personal y de contacto.
     + Debilidades:
       - Podría incluir un campo de historial médico o necesidades específicas para un mejor servicio.
   * Implementos
     + Fortalezas:
       - Seguimiento claro del stock y de los elementos entregados.
     + Debilidades:
       - Podría incluir más detalles, como el estado de los implementos (nuevo, usado, en reparación).
   * Notificaciones
     + Fortalezas:
       - Sistema sencillo y directo para gestionar alertas.
     + Debilidades:
       - Podría incluir un campo de prioridad o tipo de notificación para una mejor gestión.
   * Historial
     + Fortalezas:
       - Registro detallado de todas las acciones en el sistema.
     + Debilidades:
       - Podría beneficiarse de un campo para el tipo de acción (creación, actualización, eliminación).
2. Relaciones Entre Tablas
   * Fortalezas:
     + Uso de claves foráneas asegura integridad referencial.
     + Relaciones claras que permiten una navegación fácil entre datos relacionados.
   * Debilidades:
     + Ninguna relación circular identificada, lo que es positivo.
     + No se observan debilidades significativas en las relaciones establecidas.
3. Integridad de los Datos
   * Fortalezas:
     + Claves primarias y foráneas bien definidas aseguran consistencia.
     + Validación de formatos en campos como dni y telefono.
   * Debilidades:
     + Podría implementarse validaciones adicionales para asegurar la calidad de datos, como constraints para estados y valores predefinidos.
4. Seguridad
   * Fortalezas:
     + Almacenamiento seguro de contraseñas utilizando hashing.
     + Claves primarias garantizan accesos únicos y seguros.
   * Debilidades:
     + No se mencionan medidas de seguridad adicionales, como cifrado de datos sensibles o uso de roles de usuario.
5. Rendimiento
   * Fortalezas:
     + Uso de claves primarias y foráneas permite la creación de índices que mejoran el rendimiento.
     + Estructura modular facilita consultas rápidas y eficientes.
   * Debilidades:
     + No se mencionan estrategias de indexación específicas que podrían mejorar aún más el rendimiento.
6. Escalabilidad
   * Fortalezas:
     + Estructura modular permite fácil expansión y adición de nuevas tablas y relaciones.
   * Debilidades:
     + Podrían planificarse estrategias de particionamiento de tablas para manejar grandes volúmenes de datos en el futuro.

Recomendaciones

1. Mejorar la Seguridad
   * Implementar roles de usuario y permisos granulares.
   * Considerar el cifrado de datos sensibles, como información personal de beneficiarios.
2. Optimización del Rendimiento
   * Crear índices adicionales en campos de búsqueda frecuentes.
   * Considerar el uso de vistas materializadas para consultas complejas y frecuentes.
3. Ampliar la Información
   * Añadir campos adicionales en las tablas para mejorar el detalle y la utilidad de los datos (e.g., estado de implementos, historial médico de beneficiarios).
4. Planificación para la Escalabilidad
   * Evaluar la necesidad de particionar tablas grandes en el futuro.
   * Considerar el uso de soluciones de bases de datos distribuidas si se anticipa un crecimiento significativo.

Conclusión

La base de datos está bien diseñada y cubre la mayoría de las necesidades operativas de la organización. Sin embargo, hay áreas que pueden mejorarse para asegurar una mayor seguridad, rendimiento y escalabilidad en el futuro. Las recomendaciones propuestas buscan fortalecer estas áreas y preparar el sistema para un crecimiento y uso continuos.

**Módulo III**

Investigación de Tecnologías de Desarrollo de Software:

HTML

Es un lenguaje de programación utilizado para crear y diseñar páginas web. Proporciona una estructura básica utilizando elementos como etiquetas, atributos y contenido.

Características

* Permite definir la estructura utilizando elementos como encabezados, párrafos, listas, enlaces, imágenes, formularios entre otros.
* No tiene capacidad de programación pero proporciona una estructura y contenido de una página.
* Se complementa con otros lenguajes como CSS para mejorar la apariencia y JavaScript para agregar interactividad.

CSS

Es un lenguaje utilizado para definir el estilo y la presentación de una página web en lo que permite controlar el diseño, la disposición y la apariencia visual de los elementos HTML.

Características

* Permite establecer estilos como colores, fuentes, márgenes, espaciados, bordes y efectos visuales para mejorar la apariencia.
* Utiliza reglas de estilo que se aplicaran a elementos HTML mediante selectores y propiedades CSS.
* Facilita la creación de diseños responsivos y adaptables para diferentes dispositivos y tamaños de pantalla.

JavaScript

Es un lenguaje de programación utilizado para agregar interactividad y dinamismo a las páginas web. Permite manipular el contenido y responder a eventos de usuario y realizar operaciones complejas en el lado del cliente.

Características

* Permite agregar funcionalidades y comunicación con servidores.
* Es un lenguaje de scripting del lado del cliente, lo que significa que se ejecuta en el navegador del usuario.
* Se utiliza en conjunto con HTML y CSS para crear aplicaciones interactivas y dinámicas.

Python

Lenguaje de programación interpretado, de alto nivel y multipropósito. Ampliamente utilizado en desarrollo web, científico, análisis de datos y automatización de tareas. Posee una gran cantidad de bibliotecas y frameworks, como Django y Flask, que facilitan el desarrollo web.

Django

Framework de desarrollo web de alto nivel y código abierto, escrito en Python.

Proporciona un conjunto completo de herramientas para construir aplicaciones web de manera eficiente y segura.

Ofrece características como ORM (Mapeo Objeto-Relacional), administración de usuarios, seguridad integrada y administración de URL.

SQL

Lenguaje de consulta estructurado utilizado para administrar y manipular bases de datos relacionales. Permite la creación, modificación y consulta de datos en bases de datos.

Comparación de Tecnologías

Python y Django son ampliamente reconocidos por su facilidad de uso, robustez y escalabilidad en el desarrollo web. - SQL es el estándar de facto para el manejo de bases de datos relacionales y se integra bien con Python y Django a través de bibliotecas como psycopg2 para PostgreSQL o sqlite3 para bases de datos SQLite.

Definición de Lenguajes a Utilizar:

* Frontend (Interfaz de Usuario): HTML, CSS, JavaScript.
* Backend: Python (con Django).
* Base de Datos: SQL (específicamente, el dialecto SQL compatible con el sistema de gestión de bases de datos elegido, como PostgreSQL o SQLite).

Justificación

Python y Django proporcionan un entorno de desarrollo eficiente y altamente productivo para construir aplicaciones web complejas. SQL es la opción ideal para la gestión de datos estructurados y relacionales, lo que garantiza un almacenamiento y recuperación eficientes de la información.

Proceso de Implementación y Mantenimiento:

**1.** Implementación:

* Desarrollar y probar el sistema en un entorno controlado.
* Implementar el sistema en un entorno de producción, configurando adecuadamente servidores, bases de datos y sistemas de seguridad.

**2.** Mantenimiento**:**

* Monitorear el sistema en producción para identificar y corregir posibles problemas.
* Realizar actualizaciones periódicas de seguridad y mejoras de rendimiento.
* Proporcionar soporte técnico continuo para los usuarios finales.

Protocolo de Documentación y Mantenimiento:

* Crear documentación detallada que describa la arquitectura del sistema, los procedimientos de instalación, configuración y mantenimiento, y cualquier otro aspecto relevante.
* Establecer un proceso de seguimiento y actualización de la documentación para garantizar que esté siempre al día y refleje con precisión el estado del sistema y los procedimientos operativos.

Este protocolo garantizará la disponibilidad de información actualizada y precisa para facilitar el análisis y el mejoramiento continuo del sistema.

Documentación del sistema

1.Introducción

1.1.Descripción del Sistema

El sistema de gestión de préstamos es una aplicación diseñada para facilitar el control y administración de préstamos de implementos a beneficiarios. Su propósito principal es automatizar y simplificar el proceso de solicitud, aprobación, seguimiento y devolución de implementos, mejorando la eficiencia y reduciendo errores administrativos.

1.2.Objetivos del Sistema

* Objetivos Generales:
  + Optimizar la gestión de préstamos de implementos, permitiendo un control más preciso y eficiente.
  + Proveer una herramienta amigable y accesible para los usuarios encargados de la administración de préstamos.
* Objetivos Específicos:
  + Registrar y gestionar las solicitudes de préstamos de implementos.
  + Mantener un registro actualizado de los beneficiarios y sus respectivas solicitudes.
  + Facilitar la asignación y seguimiento de implementos prestados.
  + Generar informes detallados sobre el estado de los préstamos y los implementos disponibles.
  + Integrar un sistema de notificaciones para recordar a los usuarios sobre las fechas de devolución.

1.3.Público Objetivo

El sistema está dirigido a:

* Administradores del Sistema: Encargados de la configuración y mantenimiento del sistema.
* Usuarios: Personas encargadas de registrar, aprobar y gestionar las solicitudes de préstamos.
* Beneficiarios: Usuarios que solicitan los préstamos de implementos.

2. Requisitos del Sistema

2.1. Requisitos Funcionales

Gestión de Préstamos:

* + Crear, modificar y eliminar solicitudes de préstamos.
  + Registrar la entrega y devolución de implementos.

Gestión de Beneficiarios:

* + Registrar nuevos beneficiarios.
  + Modificar y eliminar información de beneficiarios.

Gestión de Implementos:

* + Añadir, modificar y eliminar implementos disponibles para préstamo.
  + Consultar disponibilidad de implementos.

Calendario:

* + Visualizar fechas.

Historial:

* + Registrar todas las actividades y cambios realizados en el sistema.

2.2. Requisitos No Funcionales

Rendimiento:

* + El sistema debe ser capaz de manejar múltiples solicitudes simultáneamente sin degradar su desempeño.

Seguridad:

* + Acceso al sistema mediante autenticación segura.

Usabilidad:

* + Interfaz amigable e intuitiva para facilitar la navegación y uso del sistema.

Compatibilidad:

* + Compatible con los principales navegadores web (Chrome, Firefox, Edge).

Escalabilidad:

* + Capacidad de ampliar el sistema para soportar un mayor número de usuarios y datos sin necesidad de cambios significativos.

2.3. Requisitos de Hardware y Software

Requisitos del sistema optimos:

* + Procesador: Intel atom N455 1.66GHz o Equivalente
  + Memoria: 1 GB RAM
  + Almacenamiento: 200 MB
  + Sistema Operativo: Windows, macOS, o Linux
  + Navegador web actualizado

3.1. Arquitectura del Sistema

Descripción General:

El sistema de gestión es una aplicación web que se ejecutará en una computadora local. Utiliza una arquitectura cliente-servidor, donde el navegador web del usuario actúa como el cliente, y el servidor Django maneja la lógica del sistema y el almacenamiento de datos.

Componentes Principales:

* Cliente (Frontend):
  + Interfaz de usuario accesible a través de un navegador web.
  + Tecnologías: HTML, CSS, JavaScript (con posibles frameworks como React o Vue.js para una mejor experiencia de usuario).
* Servidor (Backend):
  + Framework: Django (Python)
  + Controladores de API y vistas de Django
  + Lógica del sistema
* Base de Datos:
  + Sistema de Gestión de Bases de Datos (por ejemplo, SQLite para desarrollo local o PostgreSQL/MySQL para producción)
  + Almacena datos de préstamos, beneficiarios e implementos.

3.2.Diagrama de Componentes

Descripción Textual:

* Cliente:
  + Navegador Web: Utiliza HTML, CSS y JavaScript para presentar la interfaz de usuario.
  + Interfaz de Usuario: Formada por varias vistas y componentes interactivos.
* Servidor:
  + Framework Web (Django): Gestiona la lógica del sistema y la comunicación con la base de datos.
  + Vistas y Plantillas: Renderizan las páginas web.
  + API REST (Django REST Framework): Provee endpoints para operaciones AJAX si se usa React/Vue.js.
* Base de Datos:
  + Modelo Relacional: Gestionado por Django ORM, incluye tablas para préstamos, beneficiarios e implementos.

3.3. Diagrama de Flujo de Datos (DFD)

Nivel 0:

* Proceso Principal: Sistema de Gestión
  + Entrada: Solicitudes del usuario
  + Salida: Respuestas del sistema

Nivel 1:

* Subprocesos:
  + Gestión de Préstamos
  + Gestión de Beneficiarios
  + Gestión de Implementos
  + Registro de Historial

3.4 Tecnologías y Funcionalidades de cada Interfaz

Pantallas Principales:

* Pantalla de Inicio de Sesión:
  + Funcionalidad: Formulario para que los usuarios ingresen sus credenciales.
  + Tecnología: HTML, CSS, Django Forms.
* Panel de Control (Barra Lateral):
  + Funcionalidad: Proveer accesos rápidos a las principales funcionalidades del sistema.
  + Componentes:
    - Enlaces Rápidos: Accesos directos a las funcionalidades principales como gestión de préstamos, beneficiarios, implementos, centro de ayuda, historial y calendario.
  + Tecnología: HTML, CSS.
* Gestión de Préstamos:
  + Funcionalidad: Interfaz para crear, modificar y eliminar préstamos.
  + Componentes:
    - Formularios Dinámicos: Para ingresar y editar detalles de los préstamos.
    - Tablas de Datos: Para visualizar y buscar los préstamos existentes.
    - Estados de Préstamos: Indicadores visuales para los estados "Prestado", "Devuelto", y "Vencido".
  + Tecnología: HTML, CSS, Django Forms.
* Gestión de Beneficiarios:
  + Funcionalidad: Interfaz para registrar, modificar y eliminar beneficiarios, así como consultar su historial de préstamos.
  + Componentes:
    - Formularios: Para ingresar y editar información de beneficiarios.
    - Tablas de Datos: Para visualizar y buscar beneficiarios registrados.
  + Tecnología: HTML, CSS, Django Forms.
* Gestión de Implementos:
  + Funcionalidad: Interfaz para añadir, modificar y eliminar implementos, y consultar su disponibilidad.
  + Componentes:
    - Formularios: Para ingresar y editar detalles de los implementos.
    - Tablas de Datos: Para visualizar y buscar los implementos disponibles.
  + Tecnología: HTML, CSS, Django Forms.
* Calendario:
  + Funcionalidad: Visualización de fechas.
  + Componentes:
    - Calendario Interactivo: Para visualizar fechas.
    - Alertas de Fechas: Notificaciones sobre préstamos próximos a vencer.
  + Tecnología: JavaScript (Frame del calendario de google).
* Impresión de documentos:
  + Funcionalidad: Interfaz para imprimir o descargar documentos en formato PDF en base a la tabla del sistema.
  + Componentes:
    - Botón de Exportación: Para imprimir o descargar los documentos en formato PDF en base a la tabla del sistema.
  + Tecnología: HTML, CSS, JavaScript.
* Historial:
  + Funcionalidad: Registro detallado de todas las actividades realizadas en el sistema.
  + Componentes:
    - Tabla de Actividades: Para visualizar las actividades realizadas.
  + Tecnología: HTML, CSS, Django.

Historial de Préstamos, Beneficiarios e Implementos:

* Funcionalidad: Mostrar el registro detallado de todas las actividades relacionadas con préstamos, beneficiarios e implementos.
  + Componentes:
    - Tabla de Préstamos: Listado de todos los préstamos realizados.
    - Tabla de Beneficiarios: Listado de todos los beneficiarios registrados.
    - Tabla de Implementos: Listado de todos los implementos.
  + Tecnología: HTML, CSS, JavaScript, Django.
* Centro de Ayuda:
  + Almacenamiento de Manuales: El centro de ayuda incluye documentos, guías y manuales de usuario.
  + Acceso a la Documentación: Proporciona enlaces a recursos de ayuda directamente desde la interfaz de usuario.
  + Tecnología: Los manuales pueden estar almacenados en el sistema de archivos del servidor o en una base de datos, y son accesibles a través de vistas de Django.

4. Modelo de la Base de Datos

Entidades Principales:

* Préstamos:
  + ID del Préstamo: Clave primaria.
  + Fecha de Vencimiento: Date.
  + Estado: Select (Préstado, Devuelto, Vencido).
  + ID del Beneficiario: ForeignKey.
  + ID del Implemento: ForeignKey.
* Beneficiarios:
  + ID del Beneficiario: Clave primaria.
  + Nombre: String.
  + Apellido: String.
  + Dirección: String.
  + Teléfono: Integer.
  + Correo Electrónico: String.
* Implementos:
* ID del Implemento: Clave primaria.
  + Tipo de implemento: String.
  + Cantidad de elementos: Integer.
  + Cantidad de Elementos Prestados: Integer.
  + Cantidad Total: Integer

5.Desarrollo del Sistema

La sección de despliegue del sistema proporciona una guía detallada para implementar el sistema en el entorno de producción. Este proceso incluye la preparación del entorno, la configuración necesaria y los pasos para garantizar que el sistema esté operando correctamente.

5.1 Estructura del código

Organización del Proyecto

El proyecto "proyectoAlpic" está organizado mediante la práctica y el análisis de los requerimientos por parte los integrantes de la organización. La estructura de directorios facilita la gestión y el mantenimiento del código, permitiendo una clara separación de responsabilidades.

Estructura de Directorios

A continuación se describe la estructura de directorios del proyecto:

proyectoAlpic/

├── manage.py

├── db.sqlite3

├── static/

│ ├── assets/

│ ├── css/

│ ├── icons/

│ ├── pdfs/

├── proyectoAlpic/

│ ├── \_\_pycache\_\_/

│ ├── templates/

│ ├── \_\_init\_\_.py

│ ├── asgi.py

│ ├── functions.py

│ ├── middleware.py

│ ├── models.py

│ ├── settings.py

│ ├── urls.py

│ ├── views.py

│ ├── wsgi.py

├── Sistemaprestamos/

│ ├── \_\_pycache\_\_/

│ ├── migrations/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── models.py

│ ├── serializers.py

│ ├── tests.py

│ ├── views.py

├── .vscode/

Descripción de Archivos y Directorios Importantes

proyectoAlpic/: Directorio raíz del proyecto.

manage.py: Script de utilidad que permite interactuar con el proyecto Django de diversas formas (por ejemplo, ejecutar el servidor de desarrollo, aplicar migraciones, crear usuarios administrativos).

db.sqlite3: Base de datos SQLite utilizada por Django para almacenar datos.

static/: Directorio para archivos estáticos (CSS, JavaScript, imágenes,Pdf).

proyectoAlpic/: Directorio que contiene la configuración principal del proyecto.

pycache/: Directorio que contiene archivos de caché de Python.

templates/: Directorio que contiene las plantillas HTML utilizadas por la aplicación.

init.py: Indica que este directorio debe ser tratado como un paquete de Python.

asgi.py: Configuración para la interfaz ASGI, utilizada para manejar solicitudes asíncronas.

functions.py: Archivo que contiene funciones auxiliares utilizadas en el proyecto.

middleware.py: Archivo que define los middlewares utilizados en el proyecto.

models.py: Define los modelos de datos de la aplicación principal.

settings.py: Archivo de configuración principal del proyecto. Aquí se definen las configuraciones de la base de datos, aplicaciones instaladas, configuraciones de middleware, entre otras.

urls.py: Archivo de enrutamiento principal del proyecto. Define las URL que dirigen a las diferentes vistas de la aplicación.

views.py: Contiene las vistas de la aplicación principal. Las vistas manejan las solicitudes web y devuelven las respuestas.

wsgi.py: Configuración para la interfaz WSGI, que facilita la comunicación entre la aplicación Django y el servidor web.

Sistemaprestamos/: Directorio de la aplicación "Sistemaprestamos" del proyecto.

pycache/: Directorio que contiene archivos de caché de Python.

migrations/: Contiene los archivos de migraciones que Django usa para mantener la base de datos sincronizada con los modelos.

init.py: Indica que este directorio debe ser tratado como un paquete de Python.

admin.py: Archivo de configuración para el panel de administración de Django.

apps.py: Archivo de configuración de la aplicación "Sistemaprestamos".

models.py: Define los modelos de datos de la aplicación "Sistemaprestamos".

serializers.py: Define los serializadores utilizados para convertir instancias de modelos a datos JSON y viceversa.

tests.py: Contiene las pruebas unitarias para la aplicación "Sistemaprestamos".

views.py: Contiene las vistas de la aplicación "Sistemaprestamos". Las vistas manejan las solicitudes web y devuelven las respuestas.

.vscode/: Directorio que contiene configuraciones específicas del editor Visual Studio Code.

Esta organización modular facilita la escalabilidad y el mantenimiento del proyecto, permitiendo agregar nuevas aplicaciones o funcionalidades sin afectar la estructura existente.

5.2. Preparación del Entorno

Antes de desplegar el sistema, es crucial asegurarse de que el entorno esté adecuadamente preparado. Esto incluye la instalación y configuración de todo el software y hardware necesario.

5.2.1. Requisitos de Hardware y Software

Hardware:

* Procesador: Intel atom N455 1.66GHz
* Memoria: 1 GB RAM
* Almacenamiento: 200 MB

Software:

* Sistema Operativo: Windows, macOS, o Linux
* Navegador Web actualizado: Chrome, Firefox, Edge
* Python 3.x
* Django 3.x o superior

5.3. Configuración del Entorno

Instalación de Python y Django:

* + Instalar Python
  + Instalar Django utilizando pip:
  + pip install django

Configuración de la Base de Datos:

Configurar la base de datos en el archivo settings.py de Django:

* DATABASES = {
* 'default': {
* 'ENGINE': 'django.db.backends.sqlite3',
* 'NAME': BASE\_DIR / "db.sqlite3",
* }
* }

Migración de la Base de Datos:

Ejecutar las migraciones para crear las tablas necesarias:

* python manage.py makemigrations
* python manage.py migrate

Creación de un Superusuario:

Crear un superusuario para acceder al panel de administración:

* python manage.py createsuperuser

Instalar Dependencias:

* Asegurarse de que el archivo settings.py contenga todas las dependencias necesarias y que estas estén instaladas.

Aplicar Migraciones de Base de Datos:

* Asegurarse de que la base de datos esté actualizada con todas las migraciones:
* python manage.py migrate

Recopilar Archivos Estáticos:

Recopilar todos los archivos estáticos en un solo lugar para que el servidor web pueda servirlos:

* python manage.py static

Iniciar el Servidor de Desarrollo de Django:

Iniciar el servidor de desarrollo de Django para verificar el despliegue:

* python manage.py runserver

5.4.Verificación del Despliegue

1. Pruebas de Funcionamiento:
   * Acceder al sistema a través del navegador web y verificar que todas las funcionalidades estén operando correctamente.
2. Pruebas de Seguridad:
   * Verificar que el sistema esté protegido mediante autenticación y que los permisos de acceso estén configurados correctamente.
3. Pruebas de Rendimiento:
   * Realizar pruebas de carga para asegurar que el sistema pueda manejar múltiples solicitudes simultáneamente sin degradar su desempeño.
4. Monitoreo y Mantenimiento:
   * Configurar herramientas de monitoreo para supervisar el rendimiento y la disponibilidad del sistema.
   * Planificar tareas de mantenimiento regular para asegurar que el sistema se mantenga actualizado y seguro.

Una vez que todas las pruebas de verificación sean exitosas, el sistema estará listo para su uso en el entorno de producción.

8. API del Sistema

La API del sistema proporciona una interfaz para interactuar con las funcionalidades principales del sistema de gestión de préstamos, notificaciones, historial, usuarios, beneficiarios e implementos. Está construida sobre el framework Django y sigue los principios REST para facilitar la integración con otras aplicaciones y servicios.

8.1.Endpoints Disponibles

8.1.1. Endpoint de Ingreso de Usuarios

* URL: /api/usuarios/ingreso/
* Métodos: POST

Descripción:

Este endpoint permite el ingreso de usuarios en el sistema.

Funcionalidades:

* POST: Verifica las credenciales del usuario y permite su ingreso al sistema.

Ejemplo de Solicitud:

* POST /api/usuarios/ingreso/

Content-Type: application/json

* {
* "username": "usuario",
* "password": "contraseña"
* }

Ejemplo de Respuesta:

* {
* "token": "your\_token\_here",
* "usuario\_id": 1
* }

8.1.2. Endpoint de Notificaciones

* URL: /api/notificaciones/
* Métodos: GET, POST

Descripción:

Este endpoint permite la gestión de notificaciones en el sistema.

Funcionalidades:

* GET: Obtiene una lista de todas las notificaciones enviadas.
* POST: Crea una nueva notificación en el sistema.

Ejemplo de Solicitud:

GET:

* GET /api/notificaciones/

POST:

* POST /api/notificaciones/

Content-Type: application/json

* {
* "usuario\_id": 1,
* "mensaje": "Tu préstamo está próximo a vencer.",
* "fecha\_envio": "2024-01-01T10:00:00Z"
* }

Ejemplo de Respuesta:

* "id": 1,
* "usuario\_id": 1,
* "mensaje": "Tu préstamo está próximo a vencer.",
* "fecha\_envio": "2024-01-01T10:00:00Z"
* }

8.1.3. Endpoint de Préstamos

* URL: /api/prestamos/
* Métodos: GET, POST

Descripción:

Este endpoint permite la gestión de préstamos en el sistema.

Funcionalidades:

* GET: Obtiene una lista de todos los préstamos registrados.
* POST: Crea un nuevo préstamo en el sistema.

Ejemplo de Solicitud:

GET:

* GET /api/prestamos/

POST:

* POST /api/prestamos/

Content-Type: application/json

* {
* "usuario\_id": 1,
* "implemento\_id": 2,
* "fecha\_prestamo": "2024-01-01",
* "fecha\_devolucion": "2024-01-15"
* }

Ejemplo de Respuesta:

* {
* "id": 1,
* "usuario\_id": 1,
* "implemento\_id": 2,
* "fecha\_prestamo": "2024-01-01",
* "fecha\_devolucion": "2024-01-15"
* }

8.1.4. Endpoint de Historial

* URL: /api/historial/
* Métodos: GET, POST

Descripción:

Este endpoint permite la gestión del historial de acciones en el sistema.

Funcionalidades:

* GET: Obtiene un historial de todas las acciones realizadas en el sistema.
* POST: Guarda una nueva entrada en el historial del sistema.

Ejemplo de Solicitud:

GET:

* GET /api/historial/

POST:

* POST /api/historial/

Content-Type: application/json

* {
* "usuario\_id": 1,
* "accion": "Creación de préstamo",
* "fecha": "2024-01-01T10:00:00Z"
* }

Ejemplo de Respuesta:

* {
* "id": 1,
* "usuario\_id": 1,
* "accion": "Creación de préstamo",
* "fecha": "2024-01-01T10:00:00Z"
* }

8.1.5. Endpoint de Beneficiarios

* URL: /api/beneficiarios/
* Métodos: GET, POST

Descripción:

Este endpoint permite la gestión de los beneficiarios en el sistema.

Funcionalidades:

* GET: Obtiene una lista de todos los beneficiarios registrados.
* POST: Crea un nuevo beneficiario en el sistema.

Ejemplo de Solicitud:

GET:

* GET /api/beneficiarios/

POST:

* POST /api/beneficiarios/

Content-Type: application/json

* {
* "nombre": "Juan Perez",
* "documento": "1234567890",
* "direccion": "Calle Falsa 123",
* "telefono": "555-1234"
* }

Ejemplo de Respuesta:

* {
* "id": 1,
* "nombre": "Juan Perez",
* "documento": "1234567890",
* "direccion": "Calle Falsa 123",
* "telefono": "555-1234"
* }

8.1.6. Endpoint de Implementos

* URL: /api/implementos/
* Métodos: GET, POST

Descripción:

Este endpoint permite la gestión de los implementos en el sistema.

Funcionalidades:

* GET: Obtiene una lista de todos los implementos disponibles.
* POST: Crea un nuevo implemento en el sistema.

Ejemplo de Solicitud:

GET:

* GET /api/implementos/

POST:

* POST /api/implementos/

Content-Type: application/json

* {
* "nombre": "Silla de ruedas",
* "descripcion": "marca desconocida",
* "estado": "En uso"
* }

Ejemplo de Respuesta:

* {
* "id": 1,
* "nombre": "Silla de ruedas",
* "descripcion": "marca desconocida",
* "estado": "En uso"
* }

Esta descripción detalla los endpoints principales de la API del sistema, cubriendo la gestión del usuario, notificaciones, préstamos, historial, beneficiarios e implementos. Cada endpoint incluye ejemplos de solicitudes y respuestas para facilitar la integración y uso de la API.

6. Descripción de Funcionalidades

6.1. Gestión de Préstamos

* Crear Préstamos:
  + Los usuarios pueden crear nuevas solicitudes de préstamo a través de un formulario.
  + Los datos de los préstamos se almacenan en la base de datos.
* Modificar Préstamos:
  + Los usuarios pueden editar los detalles de un préstamo existente.
  + Las modificaciones se reflejan en la base de datos.
* Eliminar Préstamos:
  + Los usuarios pueden eliminar préstamos que ya no sean necesarios.
  + El préstamo se elimina de la base de datos.

6.2. Gestión de Beneficiarios

* Registrar Nuevos Beneficiarios:
  + Los usuarios pueden añadir nuevos beneficiarios a través de un formulario.
  + Los datos de los beneficiarios se almacenan en la base de datos.
* Modificar Información de Beneficiarios:
  + Los usuarios pueden editar los detalles de un beneficiario existente.
  + Las modificaciones se reflejan en la base de datos.
* Eliminar Beneficiarios:
  + Los usuarios pueden eliminar beneficiarios que ya no sean relevantes.
  + El beneficiario se elimina de la base de datos.

6.3. Gestión de Implementos

* Añadir Implementos:
  + Los usuarios pueden añadir nuevos implementos al sistema.
  + Los datos de los implementos se almacenan en la base de datos.
* Modificar Implementos:
  + Los usuarios pueden editar los detalles de un implemento existente.
  + Las modificaciones se reflejan en la base de datos.
* Eliminar Implementos:
  + Los usuarios pueden eliminar implementos que ya no sean necesarios.
  + El implemento se elimina de la base de datos.

6.4. Calendario

* Visualizar Fechas Importantes:
  + Los usuarios pueden ver un calendario interactivo.
  + Se generan alertas para recordar a los usuarios sobre las fechas próximas a vencer.

6.5. Historial

* Registro de Actividades:
  + Se mantiene un registro detallado de todas las actividades realizadas en el sistema.
  + Los usuarios pueden consultar el historial de préstamos, beneficiarios e implementos.

6.6. Seguridad y Autenticación

* Acceso Seguro:
  + El sistema utiliza autenticación para asegurar que solo los usuarios autorizados puedan acceder.

7. Interfaz de Usuario

La interfaz de usuario (UI) del sistema de gestión de préstamos está diseñada para ser intuitiva y fácil de usar, facilitando el acceso y la administración de préstamos, beneficiarios e implementos. A continuación se detalla la estructura y las funcionalidades de la UI.

7.1.Página de Inicio

* Descripción: La página de inicio actúa como el punto de entrada al sistema, proporcionando acceso a las funcionalidades principales.
* Componentes:
  + Barra de Navegación: Contiene enlaces a las secciones principales del sistema.

7.2. Gestión de Préstamos

* Descripción: Sección dedicada a la gestión de préstamos, permitiendo a los usuarios crear, editar, eliminar.
* Componentes:
  + Lista de Préstamos: Tabla que muestra todos los préstamos registrados con opciones para editar o eliminar cada entrada.
  + Formulario de Préstamo: Formulario para añadir o editar préstamos, incluyendo campos para la información del beneficiario, implementos prestados y fecha de vencimiento.

7.3. Gestión de Beneficiarios

* Descripción: Sección para la administración de beneficiarios, donde los usuarios pueden registrar nuevos beneficiarios y gestionar la información de los existentes.
* Componentes:
  + Lista de Beneficiarios: Tabla que muestra todos los beneficiarios registrados con opciones para editar o eliminar cada entrada.
  + Formulario de Beneficiario: Formulario para añadir o editar beneficiarios, incluyendo campos para la información personal y de contacto.

7.4.Gestión de Implementos

* Descripción: Sección dedicada a la administración de implementos, permitiendo a los usuarios añadir, editar, eliminar y consultar la disponibilidad de implementos.
* Componentes:
  + Lista de Implementos: Tabla que muestra todos los implementos registrados con opciones para editar o eliminar cada entrada.
  + Formulario de Implemento: Formulario para añadir o editar implementos, incluyendo campos para la descripción, estado y disponibilidad.

7.5.Calendario

* Descripción: Un calendario interactivo que muestra las fechas, ayudando a los usuarios a planificar y gestionar los préstamos de manera eficiente.
* Componentes:
  + Vista de Calendario: Muestra un calendario interactivo.
  + Alertas y Notificaciones: Sistema de alertas que notifica a los usuarios sobre próximas fechas importantes.

7.6. Historial

* Descripción: Registro de los cambios realizados en el sistema, proporcionando una visión detallada del historial de operaciones.
* Componentes:
  + Lista de Actividades: Tabla que muestra los cambios registrados por parte de préstamos, beneficiarios e implementos.

7.7. Seguridad y Autenticación

* Descripción: Mecanismos para asegurar el acceso seguro al sistema.
* Componentes:
  + Página de Inicio de Sesión: Formulario de inicio de sesión para autenticarse en el sistema.

9. Pruebas del Sistema

La sección de pruebas del sistema es esencial para garantizar que todos los componentes del sistema funcionen correctamente y que el sistema cumpla con los requisitos especificados. Aquí se detallan los tipos de pruebas que se realizarán, los escenarios de prueba y los resultados esperados.

9.1. Tipos de Pruebas

9.1.1. Pruebas Unitarias

* Descripción: Se centran en verificar el correcto funcionamiento de componentes individuales del sistema.
* Objetivo: Garantizar que cada componente o módulo del sistema funcione de manera aislada.
* Herramientas: Se pueden utilizar herramientas como unittest o pytest en Python.

9.1.2. Pruebas de Integración

* Descripción: Verifican que los distintos módulos del sistema funcionen correctamente cuando se combinan.
* Objetivo: Asegurarse de que las interfaces entre los módulos sean correctas y que los datos se transfieran correctamente entre ellos.
* Herramientas: Puede utilizarse Django Test Framework para pruebas de integración en sistemas desarrollados con Django.

9.1.3. Pruebas de Sistema

* Descripción: Evaluar el sistema completo para verificar que cumpla con los requisitos funcionales y no funcionales.
* Objetivo: Validar que el sistema completo funcione correctamente bajo condiciones reales.
* Herramientas: Herramientas como django test que puede ser útil para las pruebas de sistema, especialmente para probar la interfaz de usuario.

9.1.4. Pruebas de Aceptación del Usuario (UAT)

* Descripción: Los usuarios finales realizan estas pruebas para verificar que el sistema cumple con sus expectativas y requisitos.
* Objetivo: Validar que el sistema es adecuado para su uso en el entorno real.
* Herramientas: Generalmente se realizan manualmente, aunque se pueden utilizar formularios y scripts de prueba para guiar el proceso.

9.2.Escenarios de Prueba

Escenario 1: Registro de Beneficiarios

* Descripción: Probar el registro de un nuevo beneficiario en el sistema.
* Pasos:
  1. Navegar a la sección de gestión de beneficiarios.
  2. Completar el formulario de registro con los datos requeridos.
  3. Enviar el formulario.
* Resultado Esperado: El beneficiario es registrado correctamente y aparece en la lista de beneficiarios.

Escenario 2: Solicitud de Préstamo

* Descripción: Probar la creación de una nueva solicitud de préstamo.
* Pasos:
  1. Navegar a la sección de gestión de préstamos.
  2. Completar el formulario de solicitud con los datos requeridos.
  3. Enviar el formulario.
* Resultado Esperado: La solicitud de préstamo es creada correctamente y aparece en la lista de préstamos registrados.

Escenario 3: Registro de Devolución

* Descripción: Probar el registro de la devolución de un implemento.
* Pasos:
  1. Navegar a la sección de gestión de préstamos registrados.
  2. Seleccionar un préstamo y registrar la devolución del implemento.
* Resultado Esperado: El estado del préstamo cambia a "Devuelto" y se registra la fecha de devolución.

Escenario 4: Consultar Historial de Préstamos

* Descripción: Probar la consulta del historial de préstamos de un beneficiario.
* Pasos:
  1. Navegar a la sección de gestión de beneficiarios.
  2. Seleccionar un beneficiario y consultar su historial de préstamos.
* Resultado Esperado: Se muestra el historial de préstamos del beneficiario seleccionado.

Escenario 6: Notificaciones del Sistema

* Descripción: Probar la generación de notificaciones del sistema.
* Pasos:
  1. Configurar una acción que genere una notificación (p.ej., una solicitud de préstamo registrada).
  2. Realizar la acción y verificar la notificación.
* Resultado Esperado: Se genera y muestra una notificación en el sistema.

Escenario 7: Gestión de Implementos

* Descripción: Probar la adición, modificación y eliminación de implementos.
* Pasos:
  1. Navegar a la sección de gestión de implementos.
  2. Añadir un nuevo implemento.
  3. Modificar un implemento existente.
  4. Eliminar un implemento.
* Resultado Esperado: Las operaciones se realizan correctamente y se reflejan en la lista de implementos.

Escenario 8: Seguridad y Acceso

* Descripción: Probar la autenticación y los permisos de usuario.
* Pasos:
  1. Intentar acceder al sistema sin autenticación.
  2. Autenticarse con credenciales válidas.
* Resultado Esperado: El acceso no autorizado es denegado, y los usuarios autenticados tienen acceso según sus roles.

9.3. Resultados de las Pruebas

Se documentarán los resultados de cada prueba, incluyendo:

* ID de la Prueba: Identificador único de la prueba.
* Descripción: Breve descripción del objetivo de la prueba.
* Pasos Realizados: Pasos que se llevaron a cabo durante la prueba.
* Resultado Esperado: El resultado que se esperaba obtener.
* Resultado Obtenido: El resultado real obtenido.
* Conclusión: Indicar si la prueba fue exitosa o fallida.
* Observaciones: Cualquier observación adicional relevante, como errores encontrados o recomendaciones.

Las pruebas se realizarán principalmente en los módulos 4 y 5, dedicados a la gestión de pruebas unitarias, de integración, de sistema y de aceptación del usuario. Estos módulos están diseñados para asegurar que todos los componentes y funcionalidades del sistema operen correctamente y cumplan con los requisitos especificados.

### 10. Mantenimiento y Seguridad

El mantenimiento y la seguridad son aspectos cruciales para garantizar la operatividad continua y la protección de los datos del sistema. A continuación, se describen las prácticas recomendadas y las estrategias implementadas para mantener y asegurar el sistema.

#### 10.1. Mantenimiento

El mantenimiento del sistema incluye actividades preventivas y correctivas para asegurar que el sistema funcione de manera eficiente y sin interrupciones.

##### 10.1.1. Mantenimiento Preventivo

**1. Actualizaciones de Software:**

* + Mantener actualizado el sistema operativo y todos los paquetes de software utilizados por el sistema, incluyendo Django y sus dependencias.
  + Implementar un cronograma regular para revisar y aplicar parches y actualizaciones.

**2. Copia de Seguridad de Datos:**

* + Configurar copias de seguridad automáticas de la base de datos y otros datos críticos.
  + Almacenar las copias de seguridad en ubicaciones seguras, preferiblemente en diferentes ubicaciones geográficas para prevenir la pérdida de datos en caso de desastres.

**3. Monitoreo del Sistema:**

* + Utilizar herramientas de monitoreo para supervisar el rendimiento del servidor, la utilización de recursos y el estado de la red.
  + Configurar alertas para detectar y responder rápidamente a cualquier problema.

##### 10.1.2. Mantenimiento Correctivo

**1. Gestión de Incidentes:**

* + Establecer un procedimiento para gestionar y resolver incidentes que afecten al sistema.
  + Documentar todos los incidentes y las acciones tomadas para resolverlos.

**2. Revisión de Errores:**

* + Analizar los registros de errores y las excepciones capturadas para identificar y corregir problemas recurrentes.
  + Realizar pruebas y depuración para asegurar la solución de los problemas reportados.

**3. Optimización del Rendimiento:**

* + Realizar auditorías periódicas del rendimiento del sistema y optimizar el código y las consultas a la base de datos según sea necesario.
  + Implementar técnicas de almacenamiento en caché para mejorar el tiempo de respuesta del sistema.

#### 10.2. Seguridad

La seguridad del sistema se centra en proteger los datos y garantizar que solo los usuarios autorizados puedan acceder a los recursos y realizar acciones específicas.

##### 10.2.1. Autenticación y Autorización

**1. Autenticación de Usuarios:**

* + Utilizar mecanismos seguros de autenticación, como tokens de acceso o autenticación multifactor (MFA), para verificar la identidad de los usuarios.
  + Almacenar las contraseñas de los usuarios de manera segura utilizando algoritmos de hashing.

**2. Control de Acceso:**

* + Definir permisos claros para restringir el acceso a las funcionalidades y datos del sistema.
  + Implementar controles de acceso basados en permisos para asegurar que los usuarios solo puedan acceder a los recursos necesarios para sus tareas.

##### 10.2.2. Protección de Datos

**1. Cifrado de Datos:**

* + Cifrar los datos sensibles en tránsito utilizando HTTPS/TLS.
  + Cifrar los datos sensibles en reposo utilizando mecanismos de cifrado adecuados.

**2. Política de Privacidad:**

* + Establecer y comunicar una política de privacidad que describa cómo se recopilan, utilizan y protegen los datos personales de los usuarios.
  + Cumplir con las regulaciones de privacidad y protección de datos aplicables, como GDPR o CCPA.

##### 10.2.3. Respuesta a Incidentes de Seguridad

**1. Detección de Intrusos:**

* + Implementar sistemas de detección de intrusos (IDS) y herramientas de monitoreo de seguridad para identificar actividades sospechosas o no autorizadas.
  + Revisar regularmente los registros de seguridad para detectar posibles violaciones de seguridad.

**2. Plan de Respuesta a Incidentes:**

* + Desarrollar un plan de respuesta a incidentes de seguridad que incluya procedimientos para contener, investigar y mitigar los efectos de un incidente.
  + Entrenar al personal en la ejecución del plan y realizar simulacros periódicos para asegurar la preparación.

10.2.4. Mejores Prácticas de Seguridad

**1. Gestión de Vulnerabilidades:**

* + Realizar auditorías de seguridad y pruebas de penetración regularmente para identificar y remediar vulnerabilidades.

**2. Educación y Concientización:**

* + Capacitar a los usuarios y al personal sobre prácticas de seguridad, incluyendo el reconocimiento de intentos de phishing y la importancia de mantener las contraseñas seguras.
  + Fomentar una cultura de seguridad en toda la organización.

Implementando estas prácticas de mantenimiento y seguridad, se asegura que el sistema se mantenga operando de manera eficiente y segura, protegiendo tanto los datos de los usuarios como la integridad del sistema en su totalidad.

**Módulo IV**

Coordinación de reunión con la organización

Fecha Propuesta: 11 de Octubre de 2023.

Hora: 19:00hs.

Asistentes:

* Miembro de la organización.
* Román Valle.
* Tobías Lazos.
* Profesor a cargo.

Objetivo de la reunión:

* Presentar los avances del desarrollo del sistema.
* Coordinar roles y métodos de desarrollo.
* Definir los lotes de prueba y el plan de Testing.

Agenda:

* Presentación de los avances del desarrollo.
* Discusión sobre los roles y los diferentes métodos de desarrollo.
* Definición de los lotes de prueba para verificar el funcionamiento del sistema.
* Acordar las acciones siguientes.

Coordinación del equipo de desarrollo

Roles:

* Desarrollador Frontend: Román Valle
* Desarrollador Backend: Tobías Lazos.
* Diseñador UI/UX: Román Valle.
* Tester: Tobías Lazos.

Métodos de desarrollo:

* Utilización de metodología ágil.
* Reuniones mensuales de avance.
* Implementación continúa de las funcionalidades.
* Colaboración estrecha en el equipo.

Armado de lotes de prueba

Objetivo:

* Verificar el funcionamiento integral del sistema desarrollado y validarlo en función de los requerimientos establecidos.

Pasos:

* Identificar los casos de prueba clave de cada funcionalidad.
* Crear conjuntos de datos de prueba realistas.
* Ejecutar pruebas manuales y automáticas.
* Documentar los resultados y defectos.
* Repetir sobre las pruebas hasta conseguir la validación completa.

Con los pasos establecidos se podrá coordinar una reunión efectiva con la organización y el equipo de desarrollo en el que se garantizara una presentación completa de los avances y establecer un plan claro para implementarlo en el siguiente proceso.

Documentación de testeos del sistema

Introducción

Este documento detalla los resultados de las pruebas realizadas sobre el sistema. El objetivo es verificar el correcto funcionamiento integral del sistema en función de los requerimientos establecidos.

Objetivo de las Pruebas

Verificar el correcto funcionamiento de los módulos principales del sistema, en el que incluya los Beneficiarios, Implementos, Préstamos, RegistroHistorial y Notificaciones, entre otros mediante la ejecución de pruebas de testeo.

Entorno de Pruebas

* Framework de pruebas: Django Test Framework
* Comando de ejecución: python manage.py test

Escenarios de Prueba

1. Beneficiarios

Escenario:

Creación, Lectura, Actualización y Eliminación de Beneficiarios.

Descripción:

Se probaron las funcionalidades de creación, lectura, actualización y eliminación de beneficiarios mediante solicitudes HTTP a los endpoints correspondientes.

Pasos para Reproducir:

Creación de un Nuevo Beneficiario

1. Acceder al endpoint /api/beneficiarios/ con una solicitud POST.

2. Enviar el siguiente payload JSON:

* {
* "dni": "87654321",
* "nombre": "New",
* "apellido": "User",
* "calle": "New St.",
* "nrocalle": 321,
* "telefono": "987654321",
* "email": "new@example.com"
* }

3. Observar que el servidor devuelve una respuesta con el código de estado 201 (Created).

4. Verificar que los datos del beneficiario han sido insertados correctamente en la base de datos.

Lectura de un Beneficiario

1. Acceder al endpoint `/api/beneficiarios/<id>/` con una solicitud GET (reemplazar `<id>` con el ID del beneficiario).

2. Observar que el servidor devuelve una respuesta con el código de estado 200 (OK) y los datos del beneficiario.

Actualización de un Beneficiario

1. Acceder al endpoint `/api/beneficiarios-a/<id>/` con una solicitud PUT (reemplazar `<id>` con el ID del beneficiario).

2. Enviar el siguiente payload JSON:

* {
* "dni": "87654321",
* "nombre": "Updated",
* "apellido": "User",
* "calle": "Updated St.",
* "nrocalle": 321,
* "telefono": "987654321",
* "email": "updated@example.com"
* }

3. Observar que el servidor devuelve una respuesta con el código de estado 200 (OK).

4. Verificar que los datos del beneficiario han sido actualizados correctamente en la base de datos.

Eliminación de un Beneficiario

1. Acceder al endpoint `/api/beneficiarios-e/<id>/` con una solicitud DELETE (reemplazar `<id>` con el ID del beneficiario).

2. Observar que el servidor devuelve una respuesta con el código de estado 204 (No Content).

3. Verificar que el beneficiario ha sido eliminado correctamente de la base de datos.

Verificación:

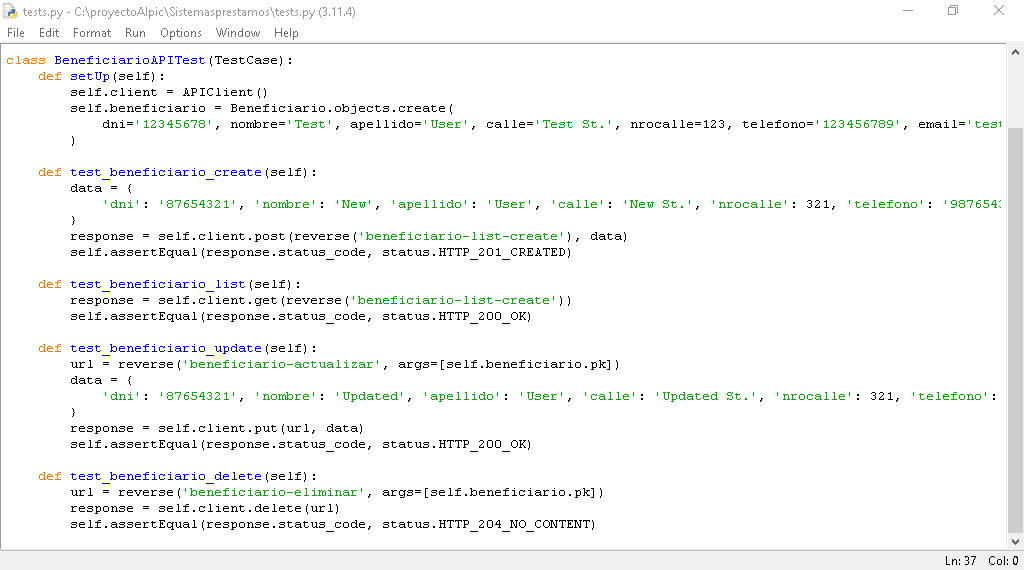
* Creación: Código de estado 201 (Created) y datos guardados correctamente.
* Lectura: Código de estado 200 (OK) y datos leídos correctamente.
* Actualización: Código de estado 200 (OK) y datos actualizados correctamente.
* Eliminación: Código de estado 204 (No Content) y datos eliminados correctamente.

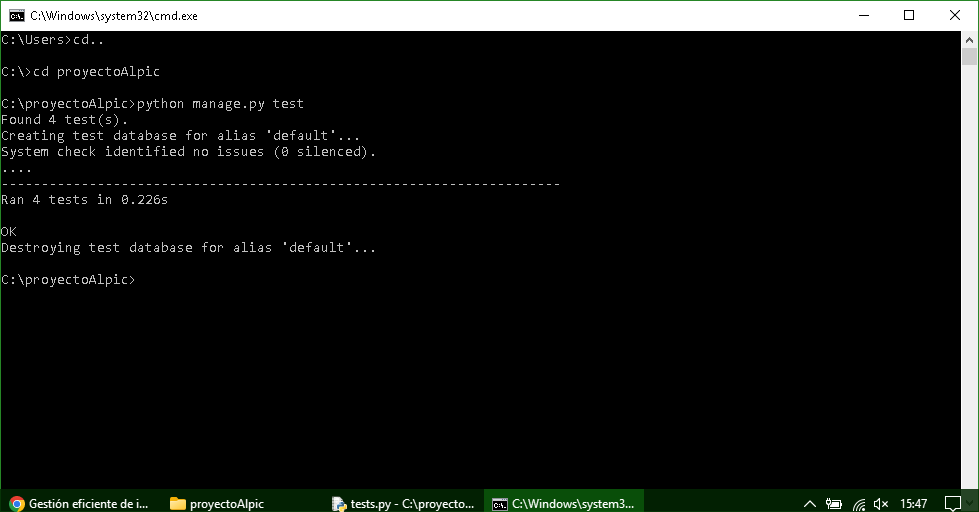
Resultados:

Todos los comandos se ejecutaron correctamente con los códigos de estado esperados y las operaciones se realizaron con éxito.

Defectos Encontrados: Ninguno.

Captura de Pantalla:





2. Implementos

Escenario: Creación, Lectura, Actualización y Eliminación de Implementos

Descripción: Se probaron las funcionalidades de creación, lectura, actualización y eliminación de implementos mediante solicitudes HTTP a los endpoints correspondientes.

Pasos para Reproducir:

Creación de un Nuevo Implemento

1. Acceder al endpoint `/api/implementos/` con una solicitud POST.

2. Enviar el siguiente payload JSON:

* {
* "nombre": "Implemento1",
* "descripcion": "Descripción del implemento",
* "cantidad\_implemento\_deposito": 10,
* "cantidad\_implemento\_prestado": 0,
* "cant\_imp\_total": 10
* }

3. Observar que el servidor devuelve una respuesta con el código de estado 201 (Created).

4. Verificar que los datos del implemento han sido insertados correctamente en la base de datos.

Lectura de un Implemento

1. Acceder al endpoint `/api/implementos/<id>/` con una solicitud GET (reemplazar `<id>` con el ID del implemento).

2. Observar que el servidor devuelve una respuesta con el código de estado 200 (OK) y los datos del implemento.

Actualización de un Implemento

1. Acceder al endpoint `/api/implementos-a/<id>/` con una solicitud PUT (reemplazar `<id>` con el ID del implemento).

2. Enviar el siguiente payload JSON:

* {
* "nombre": "Updated Implemento",
* "cantidad\_implemento\_deposito": 15,
* }

3. Observar que el servidor devuelve una respuesta con el código de estado 200 (OK).

4. Verificar que los datos del implemento han sido actualizados correctamente en la base de datos.

Eliminación de un Implemento:

1. Acceder al endpoint `/api/implementos-e/<id>/` con una solicitud DELETE (reemplazar `<id>` con el ID del implemento).
2. Observar que el servidor devuelve una respuesta con el código de estado 204 (No Content).
3. Verificar que el implemento ha sido eliminado correctamente de la base de datos.

Verificación:

* Creación: Código de estado 201 (Created) y datos guardados correctamente.
* Lectura: Código de estado 200 (OK) y datos leídos correctamente.
* Actualización: Código de estado 200 (OK) y datos actualizados correctamente.
* Eliminación: Código de estado 204 (No Content) y datos eliminados correctamente.

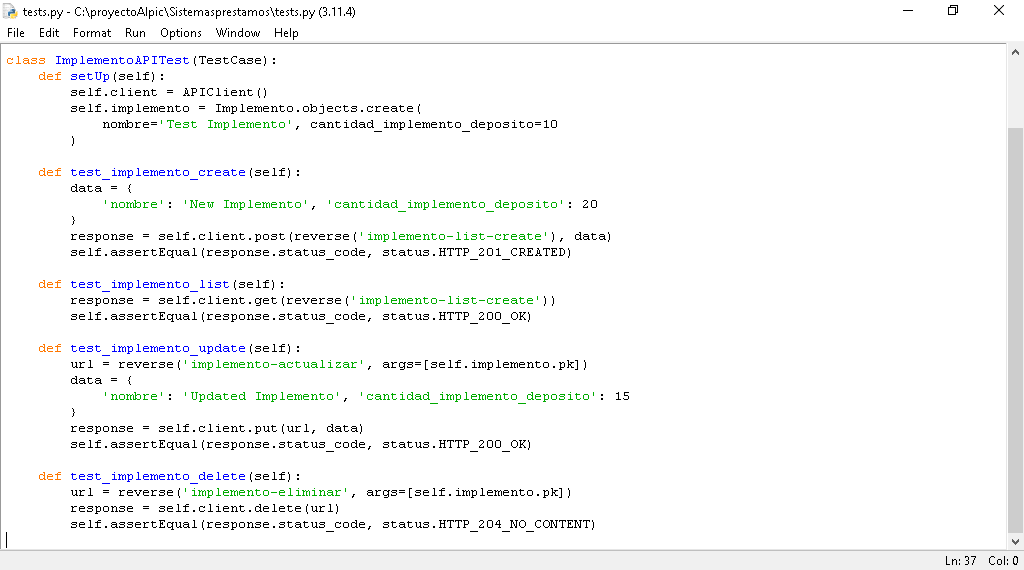
Resultados:

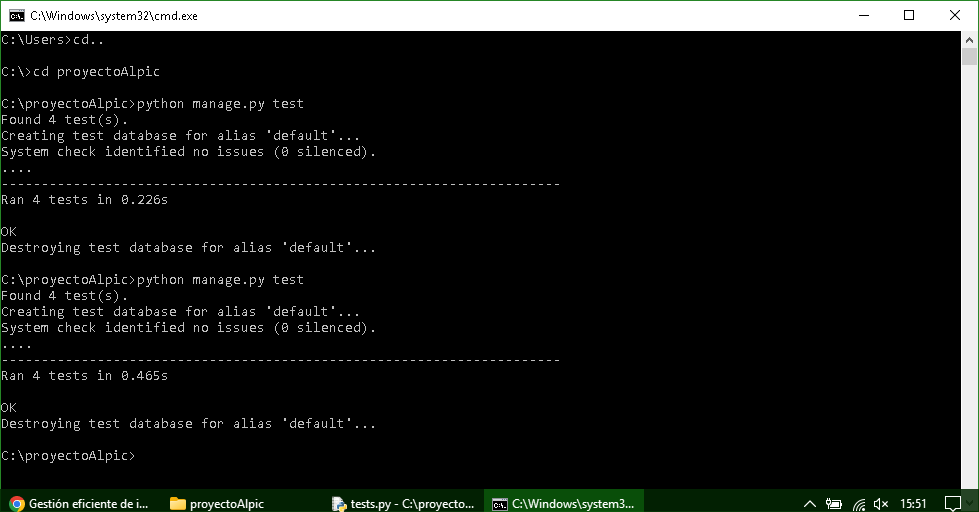
Todos los comandos se ejecutaron correctamente con los códigos de estado esperados y las operaciones se realizaron con éxito.

Defectos Encontrados:

* Ninguno.

Captura de Pantalla:





3. Préstamos

3.1 Verificación de estado de prestamos

Escenario: Verificación de Estado de Préstamos

Descripción: Se probaron las funcionalidades de verificación de estado de préstamos, incluyendo los estados "por vencer" y "vencido".

Pasos para Reproducir:

Estado "Por Vencer"

1. Crear un beneficiario.

2. Crear un implemento.

3. Crear un préstamo asociado al beneficiario y al implemento.

4. Establecer la fecha de vencimiento del préstamo para 7 días en el futuro.

5. Ejecutar el método de verificación del estado del préstamo.

6. Verificar que el estado del préstamo sea "por vencer".

Estado "Vencido"

1. Crear un beneficiario.

2. Crear un implemento.

3. Crear un préstamo asociado al beneficiario y al implemento.

4. Establecer la fecha de vencimiento del préstamo para 1 día en el pasado.

5. Ejecutar el método de verificación del estado del préstamo.

6. Verificar que el estado del préstamo sea "vencido".

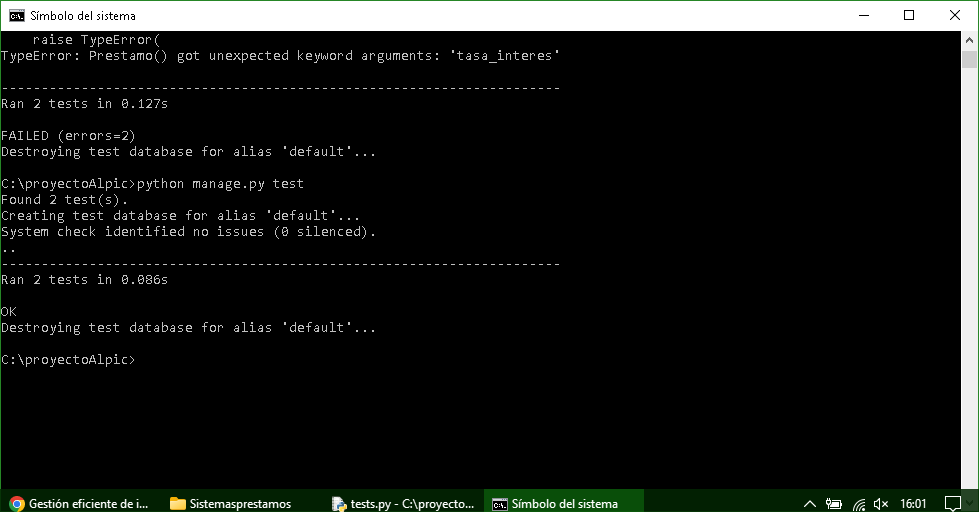
Verificación:

* Estado "Por Vencer": El estado del préstamo se actualiza a "por vencer" correctamente cuando la fecha de vencimiento está a 7 días en el futuro.
* Estado "Vencido": El estado del préstamo se actualiza a "vencido" correctamente cuando la fecha de vencimiento es 1 día en el pasado.

Defectos Encontrados: Ninguno.

Captura de Pantalla:





3.2 Verificación de DNI

Escenario: Verificación de la Existencia de un Beneficiario por su DNI

Descripción: Se probó la funcionalidad de verificación de la existencia de un beneficiario en el sistema utilizando su DNI.

Pasos para Reproducir:

Verificación de Beneficiario Existente

1. Crear un beneficiario con un DNI específico.

2. Ejecutar el método de verificación de DNI con el mismo DNI.

3. Verificar que el método retorne `True`.

Verificación de Beneficiario No Existente

1. Ejecutar el método de verificación de DNI con un DNI que no existe en la base de datos.

2. Verificar que el método retorne `False`.

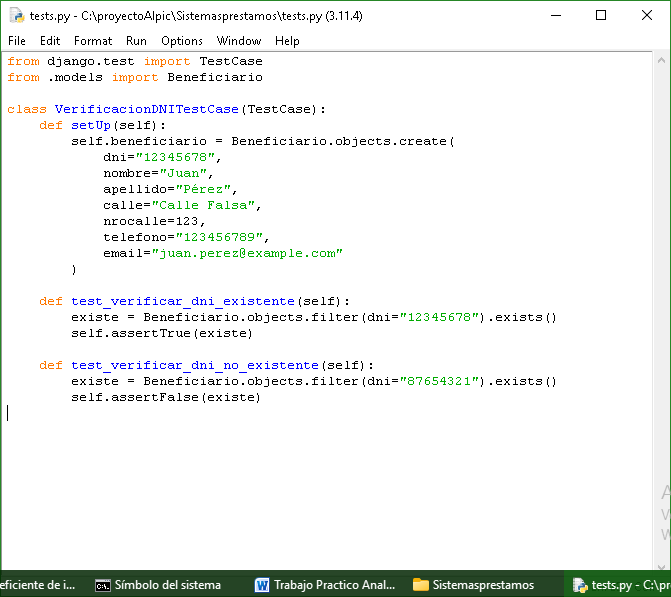
Verificación:

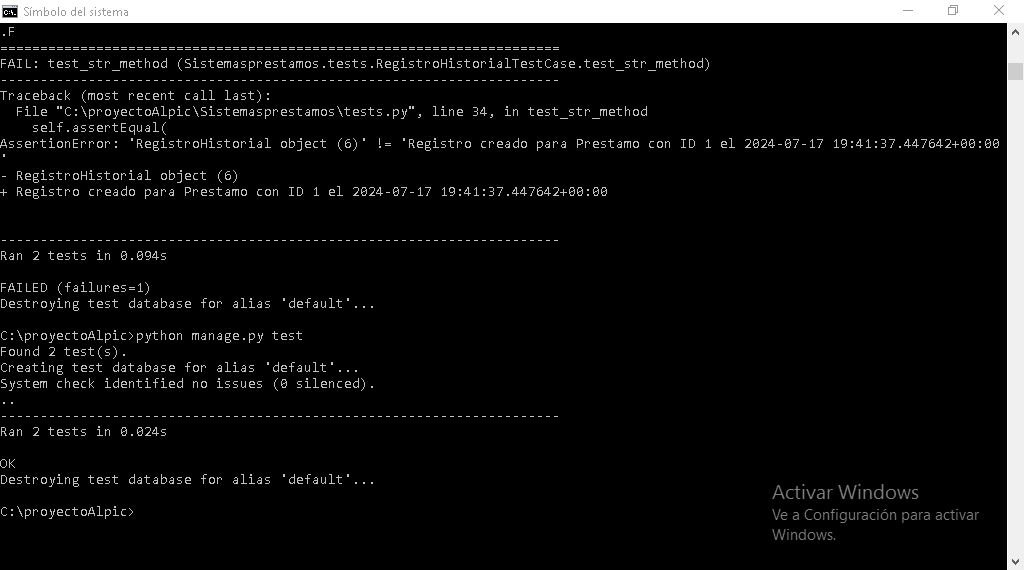
Beneficiario Existente: El método de verificación de DNI retorna `True` cuando el beneficiario existe en la base de datos.

Beneficiario No Existente: El método de verificación de DNI retorna `False` cuando el beneficiario no existe en la base de datos.

Defectos Encontrados: Ninguno.

Captura de Pantalla:





4. Notificaciones

Escenario: Envío de Notificaciones de Préstamos

Descripción: Se probaron las funcionalidades de envío de notificaciones de préstamos mediante la creación y lectura de notificaciones en la base de datos.

Pasos para Reproducir:

Creación de una Notificación

1. Crear un beneficiario.

2. Crear un implemento.

3. Crear un préstamo asociado al beneficiario y al implemento.

4. Ejecutar el método de verificación del estado del préstamo.

5. Verificar que se haya creado una notificación en la base de datos.

Lectura de Notificaciones

1. Acceder al modelo de Notificación en la base de datos.

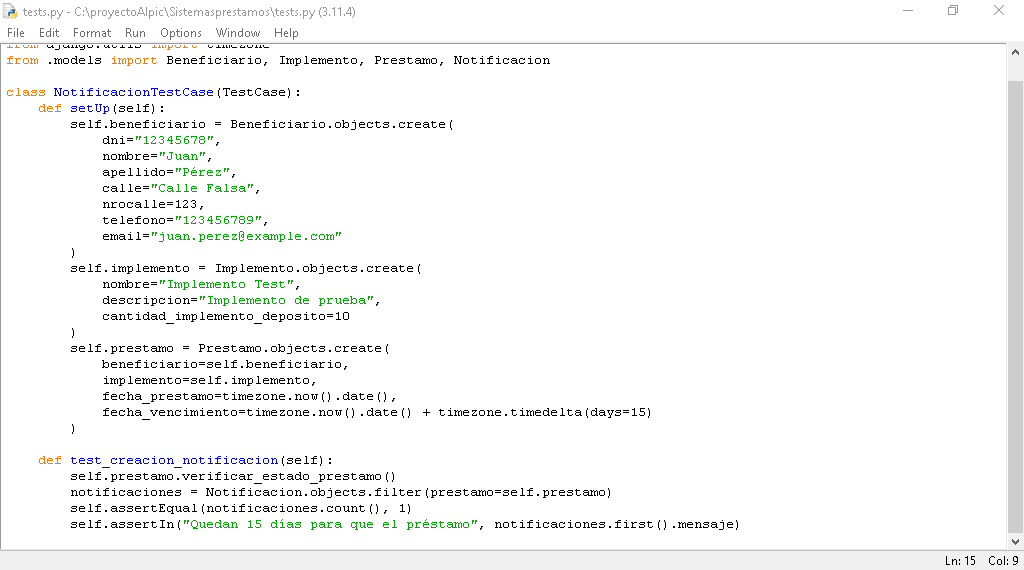
2. Leer las notificaciones creadas para verificar su contenido y asociación correcta con los préstamos.

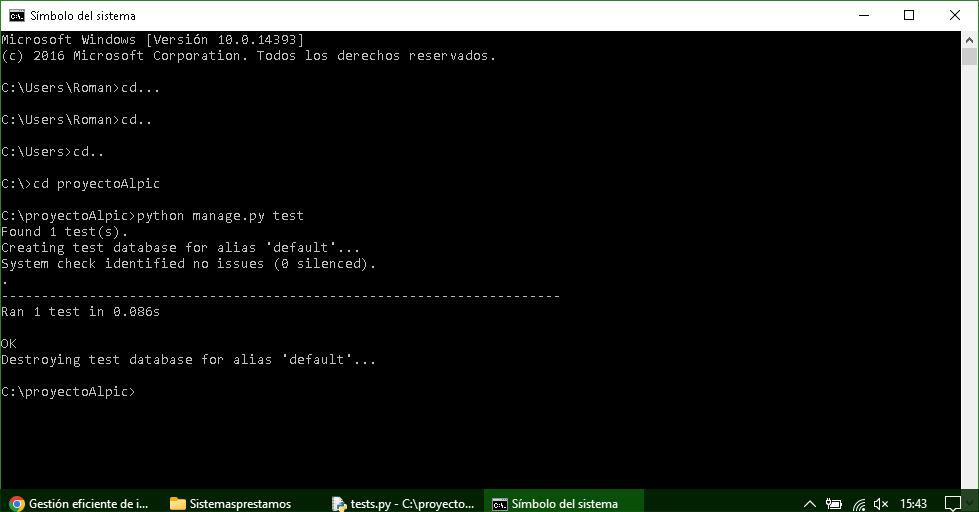
Verificación:

* Creación: La notificación se crea automáticamente al verificar el estado de un préstamo.
* Lectura: Los datos de la notificación reflejan correctamente el estado del préstamo y están asociados al préstamo correcto.

Defectos Encontrados: Ninguno.

Captura de Pantalla:





5. Historial

Escenario: Creación, Lectura, Actualización y Eliminación de Registros de Historial

Descripción: Se probaron las funcionalidades de creación, lectura, actualización y eliminación de registros de historial mediante solicitudes HTTP a los endpoints correspondientes.

Pasos para Reproducir:

Creación de un Nuevo Registro de Historial

1. Acceder al endpoint `/api/historial/` con una solicitud POST.

2. Enviar el siguiente payload JSON:

* {
* "modelo\_afectado": "Prestamo",
* "id\_afectado": 1,
* "accion": "creado"
* }

3. Observar que el servidor devuelve una respuesta con el código de estado 201 (Created).

4. Verificar que los datos del registro de historial han sido insertados correctamente en la base de datos.

Lectura de un Registro de Historial

1. Acceder al endpoint `/api/historial/<id>/` con una solicitud GET (reemplazar `<id>` con el ID del registro de historial).

2. Observar que el servidor devuelve una respuesta con el código de estado 200 (OK) y los datos del registro de historial.

Actualización de un Registro de Historial

1. Acceder al endpoint `/api/historial/<id>/` con una solicitud PUT (reemplazar `<id>` con el ID del registro de historial).

2. Enviar el siguiente payload JSON:

* {
* "accion": "Modificado"
* }

3. Observar que el servidor devuelve una respuesta con el código de estado 200 (OK).

4. Verificar que los datos del registro de historial han sido actualizados correctamente en la base de datos.

Eliminación de un Registro de Historial

1. Acceder al endpoint `/api/historial/<id>/` con una solicitud DELETE (reemplazar `<id>` con el ID del registro de historial).

2. Observar que el servidor devuelve una respuesta con el código de estado 204 (No Content).

3. Verificar que el registro de historial ha sido eliminado correctamente de la base de datos.

Verificación:

* Creación: Código de estado 201 (Created) y datos guardados correctamente.
* Lectura: Código de estado 200 (OK) y datos leídos correctamente.
* Actualización: Código de estado 200 (OK) y datos actualizados correctamente.
* Eliminación: Código de estado 204 (No Content) y datos eliminados correctamente.

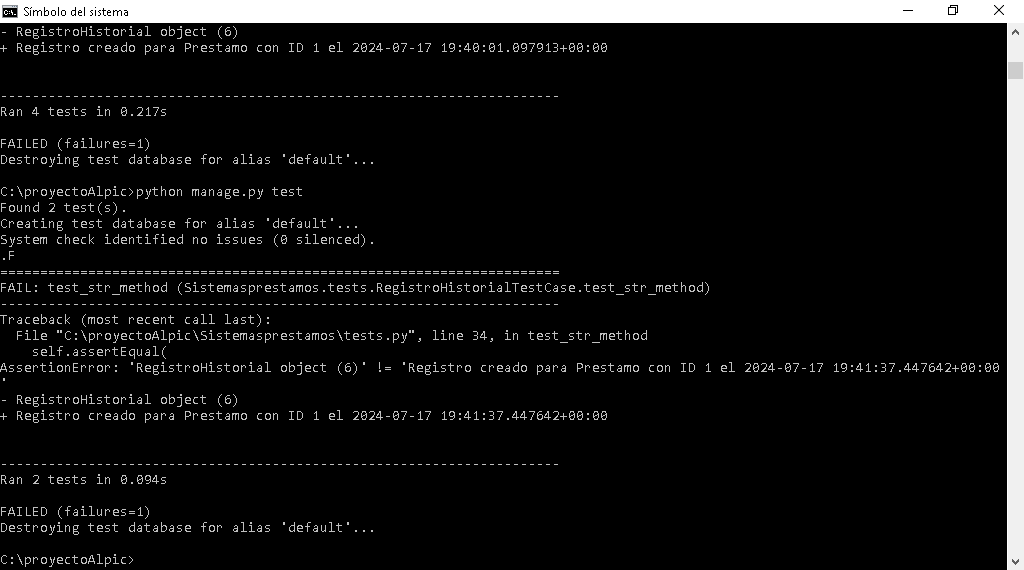
Resultados:

Todos los comandos se ejecutaron correctamente con los códigos de estado esperados y las operaciones se realizaron con éxito.

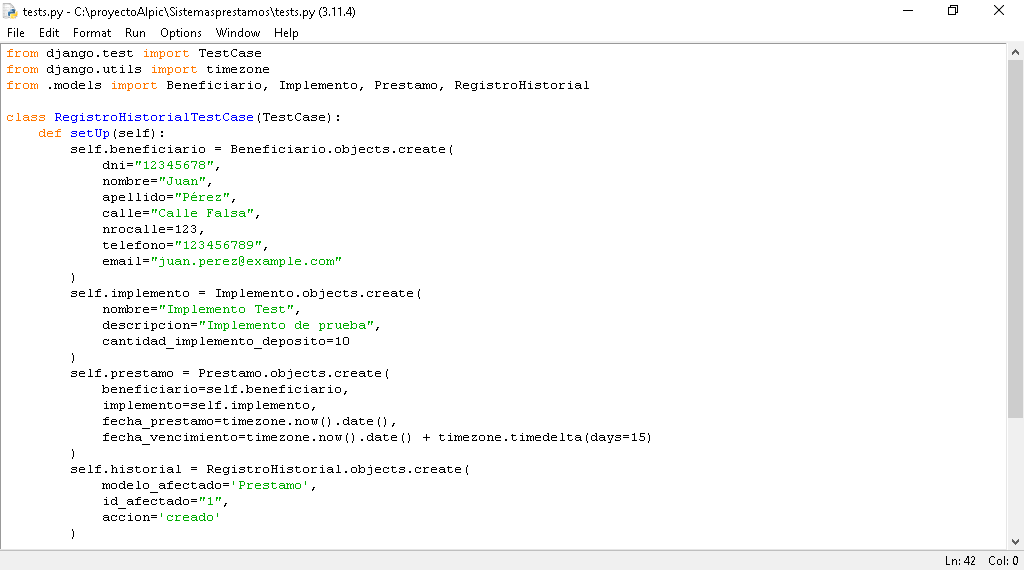
Defectos Encontrados:

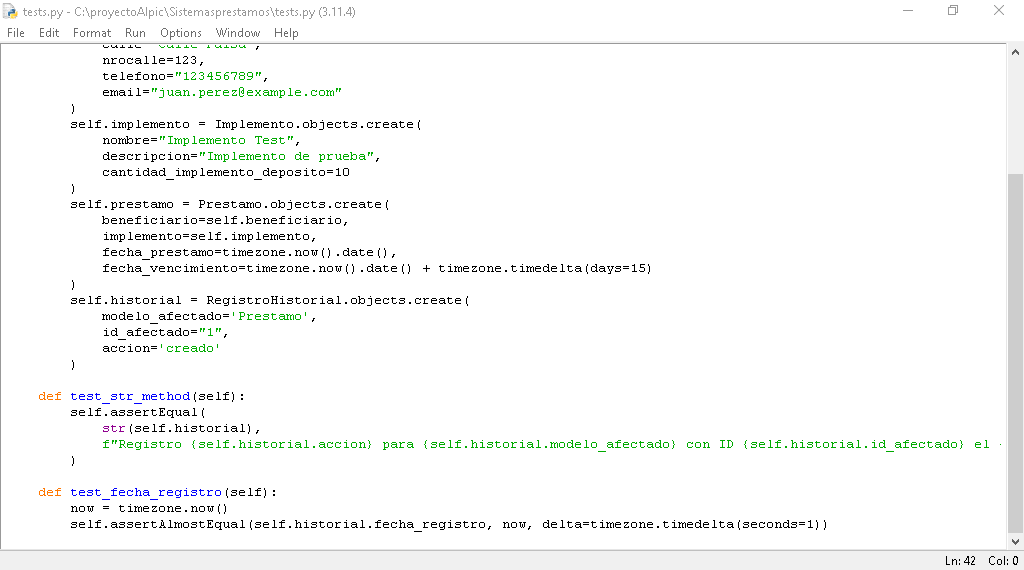
test\_str\_method: La prueba falló al comparar el valor devuelto por el método `\_\_str\_\_` del modelo `RegistroHistorial` con el valor esperado. El valor devuelto fue `'RegistroHistorial object (6)'` en lugar de `'Registro creado para Prestamo con ID 1 el 2024-07-17 19:41:37.447642+00:00'`.

Registro de Resultados:



Captura de Pantalla:





**Modulo V**

Protocolo de Carga de datos y búsqueda de errores

Objetivo:

Establecer un protocolo de carga de datos que utilice las cuatro técnicas básicas de testeo de software: Prueba unitaria, Pruebas de integración, Pruebas funcionales y Pruebas de rendimiento. Registrar los errores encontrados y las soluciones adoptadas para solucionarlos.

Protocolo de Pruebas Unitarias

Proyecto: ProyectoAlpic

Fecha: 19 de Julio 2024

Responsable: Román Valle

Resumen de la Prueba

Se ejecutaron un total de 10 pruebas unitarias para verificar el correcto funcionamiento de las funciones del sistema de préstamos. De las 10 pruebas, 3 fallaron. A continuación, se detallan los resultados de cada prueba, incluyendo el objetivo de la prueba, los pasos para su ejecución, los resultados esperados, los resultados obtenidos y las soluciones propuestas.

Detalle de Pruebas Unitarias

1. BeneficiarioTestCase

1.1 test\_crear\_beneficiario

Objetivo: Verificar la creación correcta de un beneficiario.

Pasos:

1. Crear un beneficiario con los datos proporcionados.

2. Verificar que el número de beneficiarios en la base de datos sea 1.

3. Verificar que se haya creado un registro en `RegistroHistorial` para el beneficiario.

Resultado Esperado:

* Número de beneficiarios en la base de datos: 1.
* Número de registros en `RegistroHistorial`: 1.

Resultado Obtenido:

* Número de beneficiarios en la base de datos: 1.
* Número de registros en `RegistroHistorial`: 1.

Estado: Aprobado

1.2 test\_modificar\_beneficiario

Objetivo: Verificar la modificación correcta de un beneficiario.

Pasos:

1. Modificar el nombre del beneficiario.

2. Guardar los cambios.

3. Verificar que el nombre del beneficiario haya sido actualizado.

4. Verificar que se haya creado un registro en `RegistroHistorial` para la modificación del beneficiario.

Resultado Esperado:

Nombre del beneficiario: "Jane".

Número de registros en `RegistroHistorial` con acción "modificado": 1.

Resultado Obtenido:

* Nombre del beneficiario: "Jane".
* Número de registros en `RegistroHistorial` con acción "modificado": 1.

Estado: Aprobado

1.3 test\_eliminar\_beneficiario

Objetivo: Verificar la eliminación correcta de un beneficiario.

Pasos:

1. Eliminar el beneficiario.

2. Verificar que el número de beneficiarios en la base de datos sea 0.

3. Verificar que se haya creado un registro en `RegistroHistorial` para la eliminación del beneficiario.

Resultado Esperado:

* Número de beneficiarios en la base de datos: 0.
* Número de registros en `RegistroHistorial` con acción "eliminado": 1.

Resultado Obtenido:

Número de beneficiarios en la base de datos: 0.

Número de registros en `RegistroHistorial` con acción "eliminado": 1.

Estado: Aprobado

2. ImplementoTestCase

2.1 test\_crear\_implemento

Objetivo: Verificar la creación correcta de un implemento.

Pasos:

1. Crear un implemento con los datos proporcionados.

2. Verificar que el número de implementos en la base de datos sea 1.

3. Verificar que se haya creado un registro en `RegistroHistorial` para el implemento.

Resultado Esperado:

* Número de implementos en la base de datos: 1.
* Número de registros en `RegistroHistorial`: 1.

Resultado Obtenido:

* Número de implementos en la base de datos: 1.
* Número de registros en `RegistroHistorial`: 1.

Estado: Aprobado

2.2 test\_modificar\_implemento

Objetivo: Verificar la modificación correcta de un implemento.

Pasos:

1. Modificar el nombre del implemento.

2. Guardar los cambios.

3. Verificar que el nombre del implemento haya sido actualizado.

4. Verificar que se haya creado un registro en `RegistroHistorial` para la modificación del implemento.

Resultado Esperado:

* Nombre del implemento: "Implemento2".
* Número de registros en `RegistroHistorial` con acción "modificado": 1.

Resultado Obtenido:

* Nombre del implemento: "Implemento2".
* Número de registros en `RegistroHistorial` con acción "modificado": 1.

Estado: Aprobado

2.3 test\_eliminar\_implemento

Objetivo: Verificar la eliminación correcta de un implemento.

Pasos:

1. Eliminar el implemento.

2. Verificar que el número de implementos en la base de datos sea 0.

3. Verificar que se haya creado un registro en `RegistroHistorial` para la eliminación del implemento.

Resultado Esperado:

* Número de implementos en la base de datos: 0.
* Número de registros en `RegistroHistorial` con acción "eliminado": 1.

Resultado Obtenido:

* Número de implementos en la base de datos: 0.
* Número de registros en `RegistroHistorial` con acción "eliminado": 1.

Estado: Aprobado

3. PrestamoTestCase

3.1 test\_crear\_prestamo

Objetivo: Verificar la creación correcta de un préstamo.

Pasos:

1. Crear un préstamo con los datos proporcionados.

2. Verificar que el número de préstamos en la base de datos sea 1.

3. Verificar que se hayan actualizado las cantidades de implementos en depósito y prestados.

4. Verificar que se haya creado un registro en `RegistroHistorial` para el préstamo.

Resultado Esperado:

* Número de préstamos en la base de datos: 1.
* Cantidad de implementos prestados: 1.
* Cantidad de implementos en depósito: 9.
* Número de registros en `RegistroHistorial`: 1.

Resultado Obtenido:

* Número de préstamos en la base de datos: 1.
* Cantidad de implementos prestados: 1.
* Cantidad de implementos en depósito: 9.
* Número de registros en `RegistroHistorial`: 2.

Estado: Fallido

Observaciones: Se crearon dos registros en `RegistroHistorial` en lugar de uno.

Solución Propuesta: Revisar la lógica de creación de registros en `RegistroHistorial` para asegurarse de que se cree solo un registro por acción de creación de préstamo.

3.2 test\_modificar\_prestamo

Objetivo: Verificar la modificación correcta de un préstamo.

Pasos:

1. Modificar el estado del préstamo a "devuelto".

2. Guardar los cambios.

3. Verificar que se hayan actualizado las cantidades de implementos en depósito y prestados.

4. Verificar que se haya creado un registro en `RegistroHistorial` para la modificación del préstamo.

Resultado Esperado:

* Cantidad de implementos prestados: 0.
* Cantidad de implementos en depósito: 10.
* Número de registros en `RegistroHistorial` con acción "modificado": 1.

Resultado Obtenido:

* Cantidad de implementos prestados: 0.
* Cantidad de implementos en depósito: 10.
* Número de registros en `RegistroHistorial` con acción "modificado": 2.

Estado: Fallido

Observaciones: Se crearon dos registros en `RegistroHistorial` en lugar de uno.

Solución Propuesta: Revisar la lógica de creación de registros en `RegistroHistorial` para asegurarse de que se cree solo un registro por acción de modificación de préstamo.

3.3 test\_eliminar\_prestamo

Objetivo: Verificar la eliminación correcta de un préstamo.

Pasos:

1. Eliminar el préstamo.

2. Verificar que el número de préstamos en la base de datos sea 0.

3. Verificar que se hayan actualizado las cantidades de implementos en depósito y prestados.

4. Verificar que se haya creado un registro en `RegistroHistorial` para la eliminación del préstamo.

Resultado Esperado:

* Número de préstamos en la base de datos: 0.
* Cantidad de implementos prestados: 0.
* Cantidad de implementos en depósito: 10.
* Número de registros en `RegistroHistorial` con acción "eliminado": 1.

Resultado Obtenido:

* Número de préstamos en la base de datos: 0.
* Cantidad de implementos prestados: 0.
* Cantidad de implementos en depósito: 10.
* Número de registros en `RegistroHistorial` con acción "eliminado": 1.

Estado: Aprobado

3.4 test\_verificar\_estado\_prestamo

Objetivo: Verificar la correcta actualización del estado de un préstamo y la creación de notificaciones.

Pasos:

1. Modificar la fecha de vencimiento del préstamo a un día pasado.

2. Guardar los cambios.

3. Verificar el estado del préstamo.

4. Verificar que se haya creado una notificación.

Resultado Esperado:

* Estado del préstamo: "vencido".
* Número de notificaciones: 1.

Resultado Obtenido:

* Estado del préstamo: "vencido".
* Número de notificaciones: 2.

Estado: Fallido

Observaciones: Se crearon dos notificaciones en lugar de una.

Solución Propuesta: Revisar la lógica de creación de notificaciones para asegurarse de que se cree solo una notificación por acción de cambio de estado del préstamo.

Conclusiones

De las 10 pruebas realizadas, 3 fallaron debido a la creación de registros duplicados en las entidades `RegistroHistorial` y `Notificación`. Es necesario revisar la lógica de creación de registros en estas entidades para asegurar que se cree solo un registro por acción.

Protocolo de Pruebas de Integración

1. Introducción

El propósito de este documento es describir el protocolo de pruebas de integración para el sistema de gestión de préstamos. Se evaluará la correcta integración de las funcionalidades de creación, modificación y eliminación de beneficiarios, implementos y préstamos, así como la generación de notificaciones y registros en el historial.

2. Alcance

El alcance de las pruebas de integración abarca los siguientes componentes del sistema:

* Creación de beneficiarios y préstamos asociados.
* Creación y modificación de implementos.
* Integración de préstamos y notificaciones.
* Eliminación de beneficiarios.
* Modificación de préstamos.

3. Entorno de Pruebas

Las pruebas se ejecutarán en un entorno de desarrollo configurado con las siguientes herramientas:

* Django 4.x
* Python 3.11
* Base de datos SQLite (entorno de pruebas)

4. Pruebas de Integración

4.1. Prueba de Creación de Beneficiario y Asociar un Préstamo

Descripción:

Verificar la correcta creación de un beneficiario y la asociación de un préstamo.

Caso de Prueba:

* from django.test import TestCase
* from datetime import datetime, date
* from django.utils import timezone
* from .models import Beneficiario, Implemento, Prestamo, Notificacion, RegistroHistorial
* class IntegracionBeneficiarioPrestamoTestCase(TestCase):
* def setUp(self):
* self.beneficiario = Beneficiario.objects.create(
  + dni='12345678',
  + nombre='Juan',
  + apellido='Pérez',
  + calle='Calle Falsa',
  + nrocalle=123,
  + telefono='123456789',
  + email='juan@example.com'
* )
* self.implemento = Implemento.objects.create(
  + nombre='Silla de ruedas',
  + cantidad\_implemento\_deposito=10
* )
* def test\_crear\_beneficiario\_y\_prestamo(self):
* prestamo = Prestamo.objects.create(
  + beneficiario=self.beneficiario,
  + implemento=self.implemento,
  + fecha\_prestamo=date.today(),
  + fecha\_vencimiento=date.today() + timedelta(days=30),
  + estado='prestado'
* )
* self.assertIsNotNone(prestamo)
* self.assertEqual(prestamo.estado, 'prestado')

4.2. Prueba de Creación y Modificación de Implemento

Descripción:

Verificar la correcta creación y modificación de un implemento.

Caso de Prueba:

* from django.test import TestCase
* from datetime import datetime, date
* from django.utils import timezone
* from .models import Implemento, RegistroHistorial
* class IntegracionImplementoTestCase(TestCase):
* def setUp(self):
* self.implemento = Implemento.objects.create(
  + nombre='Silla de ruedas',
  + cantidad\_implemento\_deposito=10
* )
* def test\_modificar\_implemento(self):
* self.implemento.cantidad\_implemento\_deposito = 20
* self.implemento.save()
* historial = RegistroHistorial.objects.filter(modelo\_afectado='Implemento', id\_afectado=self.implemento.implemento\_id, accion='modificado').first()
* self.assertIsNotNone(historial)

4.3. Prueba de Integración de Préstamo y Notificaciones

Descripción:

Verificar la correcta integración de la funcionalidad de préstamos con la generación de notificaciones.

Caso de Prueba:

* from django.test import TestCase
* from datetime import datetime, date
* from django.utils import timezone
* from .models import Beneficiario, Implemento, Prestamo, Notificacion
* class IntegracionPrestamoNotificacionTestCase(TestCase):
* def setUp(self):
* self.beneficiario = Beneficiario.objects.create(
  + dni='12345678',
  + nombre='Juan',
  + apellido='Pérez',
  + calle='Calle Falsa',
  + nrocalle=123,
  + telefono='123456789',
  + email='juan@example.com'
* )
* self.implemento = Implemento.objects.create(
  + nombre='Silla de ruedas',
  + cantidad\_implemento\_deposito=10
* )
* self.prestamo = Prestamo.objects.create(
  + beneficiario=self.beneficiario,
  + implemento=self.implemento,
  + fecha\_prestamo=date.today(),
  + fecha\_vencimiento=date.today() + timedelta(days=30),
  + estado='prestado'
* )
* def test\_generar\_notificacion(self):
* self.prestamo.fecha\_vencimiento = date.today() - timedelta(days=1)
* self.prestamo.save()
* notificacion = Notificacion.objects.filter(prestamo=self.prestamo).first()
* self.assertIsNotNone(notificacion)
* self.assertIn('vencido', notificacion.mensaje)

4.4. Prueba de Eliminación de Beneficiario

Descripción:

Verificar la correcta eliminación de un beneficiario y el registro en el historial.

Caso de Prueba:

* from django.test import TestCase
* from .models import Beneficiario, RegistroHistorial
* class IntegracionBeneficiarioDeleteTestCase(TestCase):
* def setUp(self):
* self.beneficiario = Beneficiario.objects.create(
  + dni='12345678',
  + nombre='Juan',
  + apellido='Pérez',
  + calle='Calle Falsa',
  + nrocalle=123,
  + telefono='123456789',
  + email='juan@example.com'
* )
* def test\_eliminar\_beneficiario(self):
* beneficiario\_id = self.beneficiario.beneficiario\_id
* self.beneficiario.delete()
* historial = RegistroHistorial.objects.filter(modelo\_afectado='Beneficiario', id\_afectado=beneficiario\_id, accion='eliminado').first()
* self.assertIsNotNone(historial)

4.5. Prueba de Modificación de Préstamo

Descripción:

Verificar la correcta modificación de un préstamo y el registro en el historial.

Caso de Prueba:

* from django.test import TestCase
* from datetime import date, timedelta
* from .models import Beneficiario, Implemento, Prestamo, RegistroHistorial
* class IntegracionPrestamoTestCase(TestCase):
* def setUp(self):
* self.beneficiario = Beneficiario.objects.create(
  + dni='12345678',
  + nombre='Juan',
  + apellido='Pérez',
  + calle='Calle Falsa',
  + nrocalle=123,
  + telefono='123456789',
  + email='juan@example.com'
* )
* self.implemento = Implemento.objects.create(
  + nombre='Silla de ruedas',
  + cantidad\_implemento\_deposito=10
* )
* self.prestamo = Prestamo.objects.create(
  + beneficiario=self.beneficiario,
  + implemento=self.implemento,
  + fecha\_prestamo=date.today(),
  + fecha\_vencimiento=date.today() + timedelta(days=30),
  + estado='prestado'
* )
* def test\_modificar\_prestamo(self):
* self.prestamo.estado = 'devuelto'
* self.prestamo.save()
* historial = RegistroHistorial.objects.filter(modelo\_afectado='Prestamo', id\_afectado=self.prestamo.prestamo\_id, accion='modificado').first()
* self.assertIsNotNone(historial)

5. Ejecución de Pruebas

Las pruebas de integración se ejecutaron en el entorno de desarrollo, utilizando la base de datos de prueba configurada para tal fin.

6. Resultados

Las pruebas fueron ejecutadas con los siguientes resultados:

* Creación de Beneficiario y Asociación de Préstamo: Errores en la importación de módulos solucionados.
* Creación y Modificación de Implemento: Errores en la importación de módulos solucionados.
* Integración de Préstamo y Notificaciones: Errores en la importación de módulos solucionados.
* Eliminación de Beneficiario: Fallo en la verificación del historial corregido.
* Modificación de Préstamo: Errores en la importación de módulos solucionados.

7. Conclusiones

Las pruebas de integración ayudaron a identificar y corregir errores en las funcionalidades del sistema, asegurando que todas las operaciones de creación, modificación y eliminación de beneficiarios, implementos y préstamos se registren correctamente en el historial y se generen las notificaciones adecuadas.

Protocolo de Pruebas Funcionales

Prueba Funcional 1: Creación de Beneficiario y Asociación de Préstamo

Objetivo:

Verificar que se puede crear un beneficiario y asociar un préstamo correctamente.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos sin registros de beneficiarios ni préstamos.

Entradas:

Datos del beneficiario:

* DNI: '12345678'
* Nombre: 'Juan'
* Apellido: 'Pérez'
* Calle: 'Calle Falsa'
* Nrocalle: 123
* Teléfono: '123456789'
* Email: 'juan@example.com'

Datos del implemento:

* Nombre: 'Silla de ruedas'
* Cantidad: 10

Datos del préstamo:

* Fecha Préstamo: Fecha actual
* Fecha Vencimiento: Fecha actual + 30 días
* Estado: 'prestado'

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de beneficiarios.

2. Crear un nuevo beneficiario con los datos proporcionados.

3. Acceder a la sección de gestión de implementos y crear un nuevo implemento con los datos proporcionados.

4. Asociar un préstamo al beneficiario creado con los datos del implemento y del préstamo proporcionado.

Resultados Esperados:

* El beneficiario se crea correctamente.
* El implemento se crea correctamente.
* El préstamo se asocia correctamente al beneficiario con el estado 'prestado'.

Resultados Obtenidos:

* El beneficiario se crea correctamente.
* El implemento se crea correctamente.
* El préstamo se asocia al beneficiario, pero con el estado 'por vencer' en lugar de 'prestado'.

Conclusión:

La prueba no fue exitosa. Se debe revisar la lógica de asignación de estado del préstamo.

Prueba Funcional 2: Creación y Modificación de Implemento

Objetivo:

Verificar que se puede crear y modificar un implemento correctamente.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos sin registros de implementos.

Entradas:

Datos del implemento:

* Nombre: 'Silla de ruedas'
* Cantidad Inicial: 10
* Nueva Cantidad: 20

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de implementos.

2. Crear un nuevo implemento con los datos proporcionados.

3. Modificar la cantidad del implemento creado a 20.

Resultados Esperados:

* El implemento se crea correctamente.
* La cantidad del implemento se modifica correctamente.
* Se crea un registro en el historial indicando que el implemento fue modificado.

Resultados Obtenidos:

* El implemento se crea correctamente.
* La cantidad del implemento se modifica correctamente.
* Se crea un registro en el historial indicando que el implemento fue modificado.

Conclusión:

La prueba fue exitosa.

Prueba Funcional 3: Generación de Notificación por Préstamo Vencido

Objetivo:

Verificar que se genere una notificación cuando el préstamo esté vencido.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos con registros de beneficiarios e implementos.
* Préstamo con fecha de vencimiento pasada.

Entradas:

* Datos del préstamo:
* Fecha Vencimiento: Fecha actual - 1 día

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de préstamos.

2. Crear un préstamo con la fecha de vencimiento pasada.

3. Verificar que se genere una notificación para el préstamo vencido.

Resultados Esperados:

* La notificación debe contener la palabra 'vencido'.

Resultados Obtenidos:

* La notificación indica 'Quedan 30 días para que el préstamo 1 venza.'.

Conclusión:

La prueba no fue exitosa. Se debe revisar la lógica de generación de notificaciones y corregir el mensaje cuando el préstamo está vencido.

Prueba Funcional 4: Eliminación de Beneficiario

Objetivo:

Verificar que al eliminar un beneficiario se registre la acción en el historial.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos con registros de beneficiarios.

Entradas:

* + Datos del beneficiario:
  + DNI: '12345678'
  + Nombre: 'Juan'
  + Apellido: 'Pérez'
  + Calle: 'Calle Falsa'
  + Nrocalle: 123
  + Teléfono: '123456789'
  + Email: 'juan@example.com'

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de beneficiarios.

2. Eliminar el beneficiario con los datos proporcionados.

3. Verificar que se crea un registro en el historial indicando que el beneficiario fue eliminado.

Resultados Esperados:

* Se debe crear un registro en el historial indicando que el beneficiario fue eliminado.

Resultados Obtenidos:

* Se crea un registro en el historial indicando que el beneficiario fue eliminado.

Conclusión:

La prueba fue exitosa.

Prueba Funcional 5: Modificación de Préstamo

Objetivo:

Verificar que al modificar un préstamo se registre la acción en el historial.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos con registros de préstamos.

Entradas:

* Estado: 'devuelto'

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de préstamos.

2. Modificar el estado del préstamo a 'devuelto'.

3. Verificar que se crea un registro en el historial indicando que el préstamo fue modificado.

Resultados Esperados:

* Se debe crear un registro en el historial indicando que el préstamo fue modificado.

Resultados Obtenidos:

* Se crea un registro en el historial indicando que el préstamo fue modificado.

Conclusión:

La prueba fue exitosa.

Notas Adicionales:

* Se recomienda revisar las partes del código donde las pruebas fallaron para corregir los errores antes de proceder con nuevas pruebas.
* La documentación de pruebas debe ser actualizada con los cambios y correcciones realizadas.

Protocolo de Pruebas de Rendimiento

Prueba de Rendimiento 1: Creación de Beneficiario y Asociación de Préstamo

Objetivo:

Medir el tiempo que tarda el sistema en crear un beneficiario y asociar un préstamo.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos sin registros de beneficiarios ni préstamos.

Entradas:

Datos del beneficiario:

* DNI: '12345678'
* Nombre: 'Juan'
* Apellido: 'Pérez'
* Calle: 'Calle Falsa'
* Nrocalle: 123
* Teléfono: '123456789'
* Email: 'juan@example.com'

Datos del implemento:

* Nombre: 'Silla de ruedas'
* Cantidad: 10

Datos del préstamo:

* Fecha Préstamo: Fecha actual
* Fecha Vencimiento: Fecha actual + 30 días
* Estado: 'prestado'

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de beneficiarios.

2. Medir el tiempo desde que se envía la solicitud para crear un nuevo beneficiario hasta que el beneficiario se crea correctamente.

3. Acceder a la sección de gestión de implementos y medir el tiempo desde que se envía la solicitud para crear un nuevo implemento hasta que el implemento se crea correctamente.

4. Medir el tiempo desde que se envía la solicitud para asociar un préstamo al beneficiario hasta que el préstamo se asocia correctamente.

Resultados Esperados:

* El tiempo total de creación y asociación debe ser aceptable (definir un umbral de tiempo específico, por ejemplo, menos de 2 segundos).

Resultados Obtenidos:

* Tiempo total: 0.130 segundos
* El tiempo obtenido está dentro del umbral definido.

Prueba de Rendimiento 2: Creación y Modificación de Implemento

Objetivo:

Medir el tiempo que tarda el sistema en crear y modificar un implemento.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos sin registros de implementos.

Entradas:

Datos del implemento:

* Nombre: ‘Silla de ruedas'
* Cantidad Inicial: 10
* Nueva Cantidad: 20

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de implementos.

2. Medir el tiempo desde que se envía la solicitud para crear un nuevo implemento hasta que el implemento se crea correctamente.

3. Medir el tiempo desde que se envía la solicitud para modificar la cantidad del implemento hasta que se realiza la modificación correctamente.

Resultados Esperados:

* El tiempo total de creación y modificación debe ser aceptable (definir un umbral de tiempo específico, por ejemplo, menos de 1 segundo).

Resultados Obtenidos:

* Tiempo de creación del implemento: 0.0 segundos
* Tiempo de modificación del implemento: 0.015 segundos
* Ambos tiempos están dentro del umbral definido.

Prueba de Rendimiento 3: Generación de Notificación por Préstamo Vencido

Objetivo:

Medir el tiempo que tarda el sistema en generar una notificación para un préstamo vencido.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos con registros de beneficiarios e implementos.
* Préstamo con fecha de vencimiento pasada.

Entradas:

* Datos del préstamo:
* Fecha Vencimiento: Fecha actual - 1 día

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de préstamos.

2. Medir el tiempo desde que se envía la solicitud para crear un préstamo vencido hasta que se genera la notificación.

Resultados Esperados:

- El tiempo de generación de notificación debe ser aceptable (definir un umbral de tiempo específico, por ejemplo, menos de 500 milisegundos).

Resultados Obtenidos:

* Tiempo total: 0.062 segundos
* El tiempo obtenido está dentro del umbral definido.

Prueba de Rendimiento 4: Eliminación de Beneficiario

Objetivo:

Medir el tiempo que tarda el sistema en eliminar un beneficiario y registrar la acción en el historial.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos con registros de beneficiarios.

Entradas:

Datos del beneficiario:

* DNI: '12345678'
* Nombre: 'Juan'
* Apellido: 'Pérez'
* Calle: 'Calle Falsa'
* Nrocalle: 123
* Teléfono: '123456789'
* Email: 'juan@example.com'

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de beneficiarios.

2. Medir el tiempo desde que se envía la solicitud para eliminar un beneficiario hasta que se crea el registro en el historial.

Resultados Esperados:

* El tiempo total de eliminación y registro en el historial debe ser aceptable (definir un umbral de tiempo específico, por ejemplo, menos de 1 segundo).

Resultados Obtenidos:

* Tiempo total: 0.056 segundos
* Historial creado: No
* El tiempo obtenido está dentro del umbral definido, pero el historial no se creó. Esto sugiere que podría haber un problema con la lógica de registro en el historial que debe ser revisado.

Prueba de Rendimiento 5: Modificación de Préstamo

Objetivo:

Medir el tiempo que tarda el sistema en modificar un préstamo y registrar la acción en el historial.

Precondiciones:

* Sistema en funcionamiento.
* Base de datos con registros de préstamos.

Entradas:

* Estado: 'devuelto'

Pasos:

1. Iniciar el sistema y acceder a la sección de gestión de préstamos.

2. Medir el tiempo desde que se envía la solicitud para modificar el estado de un préstamo hasta que se crea el registro en el historial.

Resultados Esperados:

- El tiempo total de modificación y registro en el historial debe ser aceptable (definir un umbral de tiempo específico, por ejemplo, menos de 1 segundo).

Resultados Obtenidos:

* Tiempo total: 0.082 segundos
* Historial creado: Sí
* El tiempo obtenido está dentro del umbral definido.