Universidad de las Ciencias Informáticas
Facultad de Tecnologias Interactivas

Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Stefanni de la Caridad López González

Tutores: M.Sc. Saylin Salas Hechevarria

Ing. Yoslenys Roque Hernández

La Habana, 2024

Hacer es la mejor manera de decir.
José Julian Martí Pérez.

Dedicatoria

Dedicado a mi hermano Jhon, que siga mis pasos y sea mejor de lo que fui, te amo.

Agradecimientos

Este trabajo representa más que el cierre de una etapa académica; es el resultado de un camino lleno de desafíos, aprendizajes y, sobre todo, el reflejo del apoyo incondicional de quienes siempre creyeron en mí.

A mi mamá, por ser el pilar fundamental de mi vida y a mi papá. A mis abuelos, que con su sabiduría me recordaron siempre la importancia de la perseverancia. A mi hermano, compañero de vida y cómplice en las alegrías y las dificultades, y a mi niño Anti, gracias por creer en mí.

A mis amigos del JPVCE Carlos Marx y de la universidad, con quienes compartí no solo horas de esfuerzo y estudio, sino también momentos de alegría que siempre llevaré en el corazón. A mi tutor Yoslenys, por ser más que un guía, un verdadero ejemplo de dedicación y paciencia. Gracias por cada consejo, cada enseñanza y por caminar conmigo en este proceso con compromiso y generosidad.

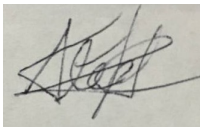
Y a mí misma, porque este camino estuvo lleno de retos, de noches interminables y de momentos en los que parecía que el esfuerzo no sería suficiente. Sin embargo, aprendí a confiar en mi y a reconocer que cada paso, por pequeño que fuera, me acercaba a la meta. Hoy puedo mirar atrás con orgullo, sabiendo que no solo superé este reto, sino que descubrí en mí una fuerza y una determinación que me llevarán más allá de lo que alguna vez imaginé. Este logro es más que un resultado, es el reflejo de mi crecimiento y de mi capacidad para seguir soñando en grande.

Gracias a todos los que fueron parte de este proceso. Este logro también es suyo.

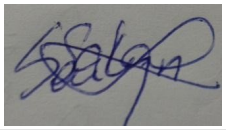
Declaración de autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

Para que así conste firmamos la presente a los 10 días del mes de diciembre del año 2024.



Stefanni de la Caridad López González
Autor



M.Sc. Saylin Salas Hechevarria
Tutor

Ing. Yoslenys Roque Hernández
Tutor

Resumen

La presente investigación tiene como objetivo el desarrollo de un Sistema de gestión de recursos y servicios para el taller de impresiones Fotoché, que mejore el control de las operaciones, superando las limitaciones del proceso manual actual. Para abordar esta problemática, se realizó un análisis de los procesos y prácticas que se siguen en el taller mediante entrevistas y observación directa. Para guiar el proceso de desarrollo, se seleccionó la metodología de Programación Extrema, la que permitió una comunicación continua con el cliente y la adaptación ágil a sus requerimientos. Se utilizó el lenguaje de Marcas de Hipertexto, TypeScript y Hojas de Estilo en Cascada para el desarrollo del frontend, proporcionándole una estructura interactiva a la propuesta. Para el desarrollo del backend, se empleó el lenguaje de programación Python junto al marco de trabajo Django, que facilitó la implementación de la lógica del negocio y la conexión con la base de datos mediante PostgreSQL. El Visual Studio Code fue la plataforma de elección, dada su versatilidad y compatibilidad con las tecnologías empleadas. La validación del sistema mediante pruebas unitarias y de aceptación, garantizó su correcto funcionamiento y una mejora en la calidad del servicio al cliente.

Palabras clave: Control, recursos y servicios, sistema de gestión, taller de impresiones.

Índice general

Índice de figuras

Índice de tablas

Introducción

En los últimos años, la introducción de las Tecnologías de la Información y las Comunicaciones (TIC) en diversos ámbitos ha modificado la estructura económica y social. Estas tecnologías han desempeñado un papel fundamental en el proceso de transformación de las dinámicas productivas al influir directamente en la competitividad de las organizaciones (Chaverra y Arias, 2012). El avance en la capacidad de procesamiento y almacenamiento de datos, junto con el desarrollo de software especializado, han facilitado la automatización de tareas que antes requerían intervención manual. Esto ha permitido a las organizaciones optimizar sus flujos de trabajo y responder de forma efectiva a las demandas del mercado (Macias y Gil, 2018).

En el contexto cubano, el avance tecnológico ha comenzado a ejercer un impacto notable en el sector empresarial, especialmente entre las pequeñas y medianas empresas. A pesar de las limitaciones económicas y estructurales que enfrenta el país, estas entidades buscan activamente optimizar sus procesos y elevar la calidad de los servicios que ofrecen. La implementación de herramientas digitales se ha convertido en una estrategia fundamental para hacer frente a los desafíos del mercado local y para satisfacer las crecientes expectativas de los consumidores, promoviendo así, una mayor eficiencia y competitividad en el entorno económico (Geographic, 2017). Esta tendencia ha impactado directamente en talleres de servicios, como los de impresión y fotografía, donde la adopción de soluciones tecnológicas ha redefinido los procesos asociados a la gestión de recursos y servicios, permitiendo responder de manera más precisa a las necesidades de los clientes (Sisternas, 2024).

En este sentido, el taller de impresiones Fotoché, ubicado en la ciudad de Cárdenas, comenzó a dar pasos en la automatización de sus tareas. En este taller se ofrecen diversos servicios orientados a la digitalización de la información como la limpieza e impresión de documentos, mecanografía, la toma, restauración y edición de fotografías, entre otros. A medida que crece la demanda se ha evidenciado que el modelo actual de operación, basado en el control manual de los procesos, presenta limitaciones que afectan directamente su capacidad de gestión. Actualmente, el registro de los de servicios ofrecidos, así como el consumo de recursos asociados, se realiza de forma manual en documentos físicos, como hojas de cálculo impresas o libretas de anotaciones. Este método no solo es ineficiente, sino también propenso a errores humanos y al extravío de información, lo que genera inconsistencias significativas en los datos operativos. Esta situación dificulta el seguimiento de las operaciones, limitando la capacidad del taller para identificar patrones de consumo, y prever necesidades futuras. Por otro lado, existe una discrepancia recurrente entre las ganancias percibidas

y las inversiones realizadas. Esta desproporción se debe, en gran medida, a la falta de un control efectivo sobre los recursos utilizados en cada servicio.

Ante la problemática anteriormente planteada surge el siguiente **problema de investigación:** ¿Cómo gestionar los recursos y servicios en el taller de impresiones Fotoché de forma tal que se contribuya a la mejora del control en las operaciones?
Se toma como **objeto de estudio** de este problema: sistemas de gestión.
La investigación se enmarca en el siguiente **campo de acción:** sistemas para la gestión de recursos y servicios.
Con el fin de solucionar la problemática planteada, se define como **objetivo general:** desarrollar un sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché de forma tal que se contribuya a la mejora del control en las operaciones.
Para dar cumplimiento al objetivo propuesto se plantean un conjunto de **objetivos específicos:**

- Identificar los referentes teóricos y metodológicos sobre el desarrollo de aplicaciones para la gestión de recursos y servicios en talleres de impresión.

- Realizar la identificación de requisitos, análisis y diseño del Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché como aproximación a la implementación.

- Implementar el Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché de forma tal que se contribuya a la mejora del control en las operaciones.

- Validar el correcto funcionamiento del Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché, aplicando técnicas y pruebas de software para contribuir a la calidad de la solución.

Para el desarrollo de la investigación se utilizaron métodos científicos, presentados a continuación:

Métodos de nivel teórico:
Los métodos teóricos cumplen una función importante, ya que posibilitan la interpretación conceptual de los datos empíricos encontrados. Así pues, los métodos teóricos al utilizarse en la construcción y desarrollo de las teorías, crean las condiciones para ir más allá de las características superficiales de la realidad, explicar los hechos y profundizar en las relaciones esenciales y cualidades fundamentales de los procesos no observables directamente (Quesada y Medina, 2020). En la presente investigación se emplearon los siguientes métodos de nivel teórico:

- **Analítico-Sintético:** se utilizó para descomponer y comprender los conceptos y aspectos relacionados con la gestión de los recursos y servicios en el taller. Además, facilitó la identificación de las herramientas, las tecnologías y el modelo apropiado para el desarrollo de la solución informática.

- **Modelación:** se empleó para representar de manera abstracta la solución propuesta, mediante la creación de modelos conceptuales y diagramas. Estas representaciones visuales ayudaron a simular el comportamiento del sistema en diferentes escenarios, lo que permitió planificar su diseño y desarrollo.

Métodos de nivel empírico:

Los métodos empíricos se basan en la experiencia en el contacto con la realidad; es decir, se fundamentan en la experimentación y la lógica que, junto a la observación de fenómenos y su análisis estadístico, son los más utilizados en el campo de las ciencias sociales y en las ciencias naturales (Quesada y Medina, 2020). En la presente investigación se emplearon los siguientes métodos de nivel empírico:

- **Entrevista:** herramienta clave para recopilar información detallada sobre el contexto permitiendo recoger datos cualitativos directamente de los involucrados, identificando con precisión las expectativas y problemas que el sistema debía resolver. Permitió capturar las percepciones de los usuarios y ajustar el diseño del sistema de acuerdo con sus demandas (Ver A.1).

- **Encuesta:** utilizada para recolectar información cuantitativa acerca de las percepciones y necesidades de los usuarios del sistema. A través de este instrumento, se logró identificar patrones y tendencias clave en la gestión de recursos y servicios, permitiendo validar las expectativas de los usuarios y establecer un marco objetivo para la toma de decisiones durante el desarrollo del sistema. Los resultados de la encuesta proporcionaron datos concretos para un mejor ajuste de la solución a las demandas específicas del cliente, contribuyendo a la mejora en la planificación y diseño del sistema (Ver A.2).

El presente trabajo está estructurado en tres capítulos:

Capítulo 1: Fundamentos y referentes teórico-metodológicos.
Este capítulo presenta una visión integral de los aspectos clave relacionados con los sistemas de gestión de recursos y servicios, y los conceptos fundamentales para su estudio. Asimismo, se lleva a cabo un análisis de sistemas homólogos para identificar mejores prácticas e implementarlas en el sistema a desarrollar. Se define la metodología a seguir, abarcando también marcos de trabajo y se exponen las herramientas y lenguajes que serán utilizados durante el desarrollo de la solución.

Capítulo 2: Planeación y diseño de la propuesta de solución.
En este capítulo se presenta una descripción de la propuesta de solución. Se aborda la disciplina de requisitos, que incluye tanto los requisitos funcionales como los no funcionales, encapsulando los primeros en historias de usuarios. Se exponen los aspectos relacionados con las etapas de planeación y diseño correspondientes a la metodología seleccionada, dando como resultado los artefactos propios de dicha metodología. Se definen los patrones de diseño y la arquitectura del sistema. Además, se realiza una estimación de esfuerzos con el objetivo de calcular la duración estimada del proceso de desarrollo.

Capítulo 3. Implementación y validación de la propuesta de solución.
Este capítulo aborda las etapas de codificación y pruebas. Se definen los estándares de codificación para garantizar la coherencia y uniformidad en el desarrollo del código. Se llevan a cabo las tareas de ingeniería correspondientes, además, se implementan pruebas unitarias y de aceptación con el objetivo de evaluar la calidad del sistema, asegurar su correcto funcionamiento y detectar posibles fallos. Estas pruebas son fundamentales para validar que la solución cumpla con los requisitos establecidos y funcione de manera correcta en diferentes escenarios de uso.

CAPÍTULO 1

FUNDAMENTOS Y REFERENTES TEÓRICO-METODOLÓGICOS

El presente capítulo tiene como objetivo establecer los fundamentos teóricos que sustentan el desarrollo del "Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché". A lo largo de este, se abordan los conceptos clave asociados al problema de la investigación. Se analiza la metodología de desarrollo seleccionada y se realiza un estudio de soluciones homólogas, tanto a nivel nacional como internacional. Además, se introducen las herramientas y tecnologías que serán utilizadas en la implementación del sistema. Este capítulo proporciona una base sólida, guiando al lector a través de los conceptos esenciales que fundamentan la solución.

1.1. Conceptos asociados al dominio del problema

A continuación, se enuncian conceptos fundamentales para ayudar al entendimiento del proyecto:

Sistema de gestión

Un sistema de gestión es una herramienta que permite controlar, planificar, organizar y, hasta cierto punto, automatizar las tareas de una empresa. Su objetivo es unificar en un software todas las operaciones de la compañía con el fin de facilitar la toma de decisiones y el análisis de los datos (Ekon, 2021).

Los sistemas de gestión, ya sea, en forma individual o integrada, deben estructurarse y adaptarse al tipo y las características de cada organización, tomando en consideración particularmente los elementos que sean apropiados para su estructuración. La implementación de un sistema de gestión eficaz permite:

- Gestionar los riesgos sociales, medioambientales y financieros.
- Mejorar la efectividad operativa.
- Reducir costos.
- Aumentar la satisfacción de clientes y partes interesadas.

- Proteger la marca y la reputación.
- Lograr mejoras continuas.
- Potenciar la innovación.
- Eliminar las barreras al comercio.
- Aportar claridad al mercado (Ekon, 2021).

Un sistema de gestión es crucial para el taller de impresiones Fotoché, ya que permite estructurar y optimizar las operaciones esenciales del negocio. Al integrar tareas como el control de recursos y servicios, además del análisis de datos, se facilita una toma de decisiones fundamentada y estratégica.

Sistema web

Un sistema web, también conocido como aplicación web, es un tipo de software que se ejecuta en un servidor remoto y se accede a través de un navegador web. A diferencia de las aplicaciones tradicionales que requieren ser descargadas e instaladas en un dispositivo, los sistemas web permiten a los usuarios interactuar con ellos a través de internet, sin necesidad de instalar ningún software adicional (Alarcon, 2024). A continuación, se listan algunas de las principales características de un sistema web:

- Accesibilidad: ofrecen acceso desde cualquier lugar con conexión a internet, lo que facilita el trabajo remoto y la colaboración entre equipos distribuidos geográficamente. Esta accesibilidad garantiza que los usuarios puedan acceder a la información y realizar tareas importantes en cualquier momento y desde cualquier dispositivo.

- Compatibilidad: una de las ventajas principales de los sistemas web es su compatibilidad con una amplia gama de navegadores web, como Google Chrome, Firefox, Safari y Edge. Esto asegura una experiencia consistente para todos los usuarios, independientemente del navegador que utilicen, lo que facilita la adopción y el uso del sistema en toda la organización.

- Costo-Efectividad: eliminan la necesidad de invertir en hardware y software costosos, ya que se ejecutan en servidores remotos y se acceden a través de navegadores web estándar.

- Escalabilidad: están diseñados para crecer con la empresa. Pueden adaptarse fácilmente a medida que aumentan las necesidades y requerimientos, ya sea agregando nuevos usuarios, funcionalidades o integraciones con otros sistemas. Esta capacidad de escalabilidad garantiza que el sistema pueda seguir siendo efectivo y útil a medida que un negocio evoluciona con el tiempo (ibíd.).

Un sistema web para la gestión de los recursos y servicios en el taller de impresiones Fotoché ofrece ventajas significativas frente a otros enfoques, debido a su naturaleza. Al ser una plataforma basada en la web, permite centralizar la información en un entorno accesible desde cualquier dispositivo conectado a la red, facilitando la supervisión en tiempo real de los recursos y servicios. La estructura modular que generalmente

caracteriza a este tipo de soluciones permite realizar ajustes y expansiones sin afectar la operatividad, lo que garantiza una gestión continua y sin interrupciones.

Control

El control es el acto de supervisar y regular diversas actividades para que se ajusten a estándares o criterios preestablecidos. Implica la evaluación constante de procesos, resultados y desviaciones con el propósito de garantizar la consecución de metas y objetivos. En el ámbito empresarial y el área de administración, el control es un mecanismo que forma parte del proceso administrativo. Su finalidad es verificar que los protocolos y objetivos de una empresa, departamento o producto cumplen con las normas y las reglas estipuladas. Como parte del proceso administrativo, el control se integra en un ciclo de retroalimentación que fomenta la mejora constante y facilita la adaptabilidad a los cambios del entorno (Obando, 2024).

En el taller de impresiones Fotoché, el control de los recursos y servicios constituye un elemento fundamental para la sostenibilidad y competitividad del negocio. Como unidad operativa, requiere supervisar de manera precisa el uso de recursos, a fin de asegurar la eficiencia en sus procesos. La naturaleza de los servicios prestados, exige una gestión organizada que permita minimizar desperdicios y garantizar la rentabilidad. Asimismo, el control favorece la capacidad del taller para planificar estratégicamente, ajustándose a las demandas del mercado y manteniendo la coherencia con los objetivos empresariales.

Recursos

Los recursos son el conjunto de factores o activos de los que dispone una empresa para llevar a cabo su estrategia (Navas Lopez y Guerras Martin, 2007). La clasificación más aceptada con respecto a los recursos de una empresa identifica la existencia de 4 tipos de recursos: Humanos, Financieros, Materiales y Técnicos (Cardenas, 2023).

- Recursos humanos: también conocido como talento o capital humano. Se refiere a los trabajadores que cumplen funciones individuales y grupales durante el proceso de producción.

- Recursos financieros: se corresponden con aquellos activos que pueden ser transformados en medios de pago sin perder valor. Son la herramienta principal tanto para el pago de salarios como para realizar inversiones por parte de la empresa, y debe llevarse una correcta gestión de ellos para el éxito del negocio.

- Recursos materiales: se constituyen a través de los muebles e inmuebles que integran el patrimonio de una empresa. Son los insumos utilizados para la elaboración de productos o servicios, que posteriormente deben llegar al cliente. Cumplen un papel importante ya que el producto final dependerá en gran parte de la calidad de éstos.

- Recursos tecnológicos: abarca todos los medios tecnológicos que son propiedad de la empresa y que permiten a los trabajadores llevar a cabo sus operaciones. Este tipo de recursos son herramientas que permiten optimizar el trabajo, encontrar o almacenar información y ahorrar tiempo (Cardenas, 2023).

En el contexto del taller de impresiones Fotoché, los recursos son indispensables para la prestación de servicios. Entre estos se incluyen:

- Tinta: esencial para el funcionamiento de impresoras, cuya calidad y propiedades técnicas influyen directamente en la claridad, el color y la durabilidad de las impresiones.

- Papel estándar: utilizado en la mayoría de los procesos de impresión, ideal para documentos cotidianos debido a su versatilidad y disponibilidad en diversos tamaños.

- Papel especializado: incluye cartulina, papel fotográfico y papel adhesivo, utilizados en trabajos que demandan mayor resistencia o una presentación específica.

Servicio

Es cualquier actividad o beneficio que una parte ofrece a otra; son esencialmente intangibles y no dan lugar a la propiedad de ninguna cosa. Su producción puede estar vinculada o no con un producto físico (Kotler y Armstrong, 1997). Existen diversas clasificaciones de los servicios, siendo una de las más comunes la distinción entre servicios tangibles e intangibles. Los servicios tangibles están vinculados a la entrega de productos físicos acompañados de un servicio adicional, como la instalación o mantenimiento de equipos. Por otro lado, los servicios intangibles se refieren a aquellos que no resultan en un producto físico, como asesoría o educación (Migallon, 2021).

En el marco del taller Fotoché, los servicios son las actividades ejecutadas por los trabajadores del taller para satisfacer las diversas necesidades de los clientes. Estos servicios abarcan una variedad de procesos, todos realizados por personal capacitado, que incluyen:

- Impresión de documentos y fotografías: consiste en la reproducción de textos e imágenes en diversos tipos de soportes, como papel común o fotográfico. La impresión puede ser en color o monocromática, dependiendo de las necesidades del cliente, y se realiza utilizando impresoras de alta calidad para garantizar resultados nítidos y precisos.

- Edición de imágenes: se realizan ajustes en fotografías o gráficos digitales, como corrección de color, mejora de resolución, recorte, o cambios en el diseño.

- Restauración de imágenes: abarca la recuperación y reparación de fotos dañadas, ya sea por el paso del tiempo, el desgaste físico o el deterioro digital. Se trabajan aspectos como la eliminación de rasgaduras, manchas o decoloraciones, y la restauración de detalles perdidos para devolver a la imagen

su apariencia original o mejorada.

- Mecanografía: se realiza la transcripción y digitalización de documentos manuscritos, viejos o deteriorados, convirtiéndolos en archivos electrónicos editables. También puede incluir la corrección de textos, adaptación de formatos o transcripción de audios a texto.

1.2. Análisis de soluciones similares

En el desarrollo de software, el análisis de sistemas homólogos es crucial para identificar soluciones previamente implementadas en contextos similares a nivel global. Además, el estudio de enfoques aplicados en diferentes entornos culturales y económicos enriquece la perspectiva del desarrollador, promoviendo la innovación y adaptación tecnológica. Como resultado, este análisis fortalece la base teórica y potencia la viabilidad del sistema propuesto.

1.2.1. Soluciones internacionales

Odoo Community Edition (CE)

Odoo Community Edition (CE) es la versión gratuita y de código abierto del software de planificación de recursos empresariales (ERP)[1] Odoo, ampliamente utilizado para la gestión de empresas y organizaciones. Al no requerir licencia, se distingue por su accesibilidad y flexibilidad, características que lo hacen ideal para pequeñas y medianas empresas que buscan una solución integral de ERP sin los costos de la versión de pago Odoo Enterprise (Technologies, 2023).

Su arquitectura modular facilita una implementación escalable y flexible, ya que permite a las empresas instalar solo los módulos necesarios para sus operaciones. Esto reduce la complejidad del sistema y optimiza el uso de los recursos disponibles. Además, está construido en el lenguaje de programación Python y utiliza el *framework* web Odoo, lo que proporciona una interfaz de usuario intuitiva y la capacidad de integración con otras herramientas empresariales y de tecnologías informáticas (ibíd.).

Odoo CE ofrece una variedad de módulos esenciales, tales como gestión de inventarios, ventas, contabilidad, recursos humanos y marketing. A diferencia de la versión Enterprise, esta edición tiene algunas limitaciones en cuanto a funcionalidades avanzadas, soporte técnico y actualizaciones de versiones más recientes. Sin embargo, cuenta con una comunidad activa que desarrolla y mantiene una amplia variedad de módulos adicionales, permitiendo a las empresas personalizar el sistema según sus necesidades específicas (Tipalti,

[1]Un ERP (*Enterprise Resorce Planing* o Planificación de recursos empresariales) es un software que permite a las empresas controlar todos los flujos de información que se generan en cada ámbito de la organización (MeyerDelius, 2022).

2023).

ERPNext

ERPNext es un software de planificación de recursos empresariales (ERP) que destaca por ser gratuito y de código abierto. Desarrollado por Frappe Technologies, este sistema está diseñado para ayudar a las organizaciones a gestionar sus operaciones de manera eficiente y efectiva. Una de sus características más notables es su amplia gama de módulos que abarcan diversas áreas funcionales dentro de una empresa. Desde contabilidad y gestión de relaciones con clientes (CRM)[2] hasta ventas, compras, inventario, recursos humanos y gestión de proyectos, ERPNext se presenta como una solución integral para empresas de diferentes tamaños y sectores. Esta diversidad permite a las organizaciones optimizar sus procesos internos y mejorar la colaboración entre departamentos (Rootstack, 2024).

La flexibilidad y personalización son dos aspectos que hacen que ERPNext sea especialmente atractivo para los usuarios. Permite adaptar sus módulos y formularios a las necesidades específicas de cada empresa, lo que facilita la alineación del sistema con los procesos internos sin requerir conocimientos avanzados en programación. Esta capacidad de personalización asegura que cada organización pueda utilizarlo como una herramienta que realmente se ajuste a su forma de trabajar (EspacioERP, 2024).

La escalabilidad del software permite a las empresas crecer y adaptarse a nuevas demandas sin necesidad de migrar a otro sistema, lo que representa una ventaja significativa en un entorno empresarial dinámico. Su naturaleza de código abierto no solo reduce costos, sino que también fomenta una comunidad activa que contribuye al desarrollo continuo del software, asegurando así su relevancia y eficacia en el futuro (ibíd.).

Dolibarr

Dolibarr es un software de planificación de recursos empresariales (ERP) y gestión de relaciones con clientes que ha ganado popularidad por su naturaleza de código abierto y su enfoque en la facilidad de uso. Desde su creación en abril de 2002, ha evolucionado significativamente, convirtiéndose en una herramienta integral para pequeñas y medianas empresas (PYMES) que buscan optimizar sus procesos administrativos y comerciales. El desarrollo de Dolibarr comenzó con la intención de proporcionar una solución accesible y flexible para la gestión empresarial (D. Foundation, 2023).

Una de las características más atractivas de Dolibarr es su flexibilidad. Los usuarios pueden habilitar solo las funciones que necesitan, lo que permite una personalización acorde a los requerimientos específicos de cada negocio. Además, es compatible con múltiples plataformas, incluyendo Windows, macOS y diversas

[2]CRM: (*Customer Relationship Management* o Gestión de relaciones con el cliente) es un software que permite a las empresas rastrear cada interacción con los usuarios y clientes actuales (Clavijo, 2024)

distribuciones de Linux, lo que facilita su implementación en diferentes entornos. La interfaz de usuario es intuitiva y está diseñada para ser amigable, lo que permite a los usuarios navegar fácilmente a través del sistema sin necesidad de formación técnica extensa. Esto es especialmente beneficioso para las PYMES que pueden no contar con personal especializado en tecnología (D. Foundation, 2023).

Dolibarr también se destaca por su accesibilidad. Al ser un software de código abierto, está disponible gratuitamente bajo la Licencia Pública General GNU. Esto significa que cualquier empresa puede descargarlo, instalarlo y utilizarlo sin costos adicionales, lo que representa una opción económica frente a soluciones comerciales más costosas (ibíd.).

1.2.2. Soluciones nacionales

VERSAT Sarasola

VERSAT Sarasola es un sistema de gestión empresarial desarrollado por DESOFT, una empresa cubana con más de dos décadas de experiencia en soluciones tecnológicas. Este software está diseñado para automatizar y optimizar los procesos administrativos y operativos de organizaciones de diversos sectores, con un enfoque modular que permite implementar únicamente las funcionalidades requeridas por cada entidad. Entre sus principales funciones se destacan la gestión de recursos humanos, que incluye control de nóminas, asistencia y capacitaciones; el manejo de inventarios y almacenes para garantizar el control y la disponibilidad de materiales; la gestión financiera y contable, adaptada a las normativas fiscales cubanas, operando tanto en pesos cubanos (CUP) como en moneda libremente convertible (MLC); y la planificación y seguimiento de órdenes de trabajo, facilitando el control de servicios. Este sistema cumple con las normativas locales, asegurando que las empresas puedan operar de manera eficiente y legal en el entorno empresarial cubano (Datazucar, 2024).

El diseño de VERSAT Sarasola está basado en tecnologías modernas que garantizan escalabilidad, rendimiento y confiabilidad, además de una alta interoperabilidad que permite su integración con otros sistemas empresariales. Su interfaz gráfica intuitiva facilita su uso tanto para técnicos como para usuarios no especializados, promoviendo una rápida adopción. Con robustos controles de acceso y gestión de usuarios, el sistema asegura la protección de los datos empresariales. DESOFT, responsable de su desarrollo, ofrece soporte técnico continuo, actualizaciones periódicas y capacitación para los usuarios, lo que asegura un alto nivel de satisfacción (ibíd.).

RECOM+ Software para la gestión comercial e inteligencia de negocios

RECOM+ es un software integral diseñado para la gestión de procesos comerciales e inteligencia de negocios, desarrollado por la Empresa de Tecnologías de la Información (ETI) en Cuba. Este sistema permite a

las organizaciones recopilar, integrar y analizar datos comerciales, facilitando así la toma de decisiones informadas y eficientes. Implementado bajo un marco regulatorio específico, RECOM+ se basa en tecnologías web y de código abierto, lo que garantiza su adaptabilidad y escalabilidad en diversas instituciones, desde las que operan en el sector biofarmacéutico hasta aquellas involucradas en comercio exterior. Su diseño modular permite la automatización de procesos que abarcan desde la gestión de compras hasta el seguimiento de contratos y proveedores (RECOM+, 2024).

Entre las funcionalidades más destacadas de RECOM+ se encuentran la gestión de carteras de clientes y proveedores, el control de información técnica relacionada con productos, y la evaluación del desempeño comercial. El software también facilita la generación de reportes analíticos que permiten a los gestores evaluar el rendimiento de las operaciones comerciales. Además, su capacidad para monitorear y evaluar cada etapa del proceso comercial asegura una trazabilidad efectiva de la información, lo que se traduce en una mejora significativa en la eficiencia operativa y en la reducción de tiempos de respuesta ante situaciones comerciales complejas (ibíd.).

1.2.3. Conclusiones del estudio de soluciones similares

Después de plantear las características de varios sistemas relacionados con el objeto de estudio, se identifican ciertos indicadores clave que permiten evaluar la viabilidad de su integración en la solución que se pretende desarrollar. Estos indicadores consideran tanto las fortalezas como las debilidades de los sistemas evaluados, con el propósito de analizar su impacto potencial en la propuesta. A continuación, se describen los parámetros seleccionados:

- Disponibilidad de código: se refiere a un modelo de desarrollo de software en el que el código fuente está disponible públicamente. Esto significa que cualquier persona puede ver, modificar y distribuir el código de acuerdo con los términos de la licencia asociada. A diferencia del software propietario, el código abierto permite a los usuarios colaborar y mejorar el programa continuamente, promoviendo la innovación y la transparencia (Stackscale, 2021).

- Gestión de recursos: se refiere a un conjunto de prácticas y herramientas que ayudan a las empresas a aprovechar al máximo sus recursos, como el tiempo, el dinero y las personas. El objetivo de la gestión de recursos es garantizar que los recursos adecuados estén disponibles en el momento adecuado para la tarea o proyecto adecuado (Clockify, 2023).

- Gestión de servicios: constituye un enfoque sistemático para planificar, diseñar y gestionar las operaciones de servicios, centrándose en satisfacer las necesidades del cliente a través de la calidad y la eficiencia operativa (Zendesk, 2023).

- Curva de aprendizaje: describe la relación entre el tiempo o el esfuerzo invertido en el aprendizaje de una tarea y el nivel de dominio alcanzado. Se fundamenta en la idea de que la adquisición de habilidades o conocimientos sigue un patrón predecible: en las primeras etapas, el progreso suele ser rápido, mientras que en etapas posteriores se ralentiza a medida que se alcanza un mayor nivel de competencia (Nunnez, 2022).

Tabla 1.1. Comparación de sistemas homólogos.

Sistemas	**Disponibilidad de código**	**Gestión de servicios**	**Gestión de recursos**	**Curva de aprendizaje**
Odoo Community Edition	Sí	Completa	Completa	Alta
ERPNext	Sí	Completa	Completa	Alta
Dolibarr	Sí	Básica	Limitada	Media
VERSAT Sarasola	No	Completa	Completa	Media
RECOM+	Sí	Completa	Básica	Media

Luego del análisis realizado a los sistemas homólogos, se determina que varios de ellos cuentan con características y funcionalidades que cubren de manera eficiente las necesidades generales de gestión empresarial. Sistemas como Odoo CE y ERPNext, destacan por su robustez y capacidad de personalización, lo que los posiciona como herramientas ampliamente aplicables en entornos diversos. Sin embargo, están diseñados para un contexto más amplio y estructurado, orientados a empresas que operan a gran escala o que requieren cubrir procesos complejos y variados.

Por otro lado, sistemas nacionales como VERSAT Sarasola y RECOM+, presentan un enfoque más regional, siendo adecuados para pequeñas y medianas empresas en mercados locales. Sin embargo, estos sistemas, aunque útiles en contextos específicos, no siempre ofrecen la personalización necesaria para satisfacer plenamente las particularidades del taller de impresiones Fotoché. Se concluye que, si bien los sistemas analizados ofrecen referencias significativas y aportan perspectivas valiosas, resulta necesario diseñar una solución adaptada a las características del taller. La implementación de un sistema personalizado no solo permitirá satisfacer los requerimientos actuales de manera precisa, sino también optimizar los procesos mediante un enfoque contextualizado y alineado con las dinámicas propias del negocio.

1.3. Metodologías de desarrollo de software

Las metodologías de desarrollo de software son un conjunto de técnicas y métodos organizativos que se aplican para diseñar soluciones de software informático. El objetivo de estas metodologías organizar los equipos de trabajo para que estos desarrollen las funciones de un programa de la mejor manera posible (Santander Open Academy, 2024).

En la actualidad se pueden diferenciar dos grandes grupos de metodologías de desarrollo de software: las ágiles y las tradicionales. Las metodologías de desarrollo de software tradicionales se caracterizan por definir total y rígidamente los requisitos al inicio de los proyectos de ingeniería de software. Los ciclos de desarrollo son poco flexibles y no permiten realizar cambios, al contrario que las metodologías ágiles; lo que ha propiciado el incremento en el uso de las segundas (Santander Open Academy, 2024).

Figura 1.1. Representación del ciclo de vida de las Metodologías tradicionales [Tomado de: (Santander Open Academy, 2024)].

Por otra parte, las metodologías ágiles se basan en la metodología incremental, en la que cada ciclo de desarrollo se va agregando nuevas funcionalidades a la aplicación final (ibíd.). Se caracterizan por su alta agilidad y flexibilidad en los procesos de trabajo. Gracias a su implementación, los equipos logran un desempeño más productivo, ya que tienen claridad sobre sus tareas y mantienen una comunicación fluida entre ellos. Además, estas metodologías se adaptan de manera óptima a los imprevistos y las necesidades que surgen a lo largo del proceso, ya que los ciclos se reducen y se vuelven significativamente más cortos (Domain Logic, 2022).

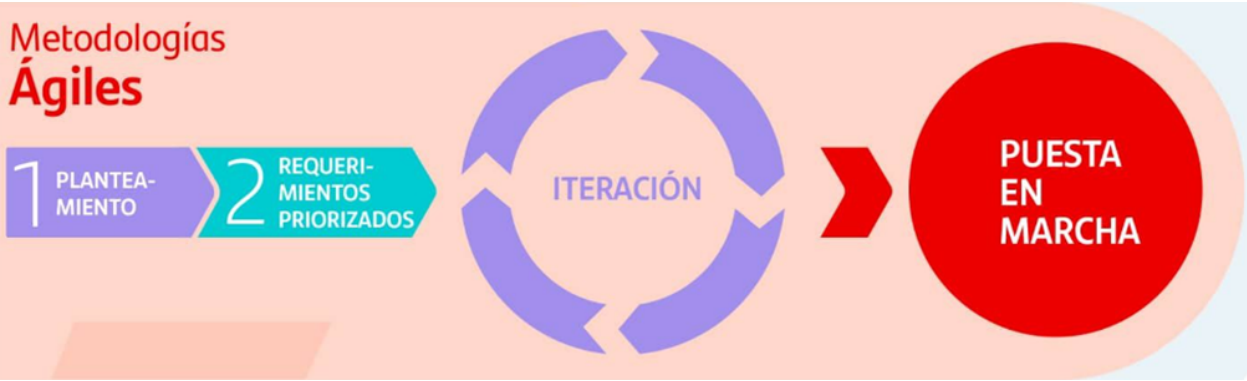


Figura 1.2. Representación del ciclo de vida de las Metodologías ágiles [Tomado de: (Santander Open Academy, 2024)].

Para guiar el desarrollo de la actual propuesta se decide optar por una metodología ágil que permita priorizar la satisfacción del cliente. Este enfoque posibilita la adaptación a cambios en los requisitos técnicos, incluso en fases avanzadas del desarrollo. Al implementar plazos definidos y entregas iterativas, se optimizan los tiempos de producción, facilitando la creación de versiones mejoradas del producto. Además, la interacción constante entre el equipo de desarrollo y el cliente proporciona una retroalimentación continua, lo que resulta esencial para garantizar la calidad del software final.
Algunas de las metodologías ágiles más utilizadas en la actualidad para el desarrollo de software son:

Kanban
Kanban es una metodología de gestión de proyectos que se centra en la visualización del flujo de trabajo y la mejora continua. Su origen se encuentra en el sistema de producción Toyota, diseñado para optimizar los procesos de fabricación mediante un control visual eficiente y basado en principios de justo a tiempo. Actualmente, se aplica en diversos sectores, especialmente en el desarrollo de software y proyectos tecnológicos (Martins, 2024).

Esta metodología utiliza un tablero visual, conocido como *Kanban Board*, que está dividido en columnas que representan las diferentes etapas del proceso de trabajo. Las tareas individuales se representan como tarjetas que se mueven a través del tablero, permitiendo a los equipos monitorear el progreso, identificar cuellos de botella y gestionar la carga de trabajo (ibíd.).

Scrum
Scrum es un marco de trabajo ágil diseñado para gestionar proyectos complejos mediante la colaboración, la adaptabilidad y la entrega iterativa de valor. Scrum se organiza en ciclos de trabajo cortos y predecibles llamados *sprints*, que generalmente tienen una duración de entre una y cuatro semanas. Cada *sprint* tiene como objetivo entregar un incremento del producto que sea funcional y tenga valor tangible (Schwaber y Sutherland, 2022).

Entre los eventos clave se encuentran la Reunión de Planificación del Sprint, donde se definen los objetivos del sprint; las Reuniones Diarias, que permiten sincronizar al equipo y ajustar el enfoque según sea necesario; la Revisión del Sprint, para evaluar los resultados entregados; y la Retrospectiva del Sprint, donde el equipo identifica oportunidades de mejora. Se basa en principios como la transparencia, la inspección y la adaptación, que garantizan la capacidad de respuesta ante cambios en los requisitos o el entorno del proyecto (ibíd.).

Programación Extrema (XP)
Programación Extrema (XP) es una metodología ágil diseñada para mejorar la calidad del software y la capacidad de respuesta a los cambios en los requisitos del cliente. Su enfoque se basa en prácticas técni-

cas intensivas y colaborativas, promoviendo la entrega frecuente de pequeñas iteraciones funcionales (Beck y Andres, 2004).

Se caracteriza por incorporar prácticas fundamentales como la programación en parejas (*pair programming*), el desarrollo guiado por pruebas (*test-driven development*, TDD), la integración continua y la propiedad colectiva del código. Estas prácticas buscan minimizar errores y garantizar que el sistema evolucione de manera sostenible (ibíd.).

Tabla 1.2. Comparación entre las metodologías Kanban, Scrum y XP.

Características	Kanban	Scrum	XP
Propósito	Optimizar el flujo de trabajo mediante la visualización y la gestión continua.	Gestión ágil de proyectos mediante ciclos iterativos y entregas incrementales.	Mejora de la calidad del software a través de prácticas técnicas avanzadas.
Estructura	Flujo continuo sin iteraciones predefinidas; uso de tableros Kanban.	Sprints definidos con duración fija (1-4 semanas); roles específicos.	Iteraciones cortas con enfoque técnico en cada ciclo.
Roles principales	No define roles específicos; se adapta a cualquier equipo.	Product Owner, Scrum Master y Equipo de Desarrollo.	Equipo de Desarrollo; cliente involucrado activamente.
Enfoque de planificación	Basado en la gestión de tareas en curso (WIP).	Planificación al inicio de cada sprint.	Iteraciones planificadas; cambios aceptados en tiempo real.
Flexibilidad ante cambios	Alta; acepta cambios en cualquier momento del flujo de trabajo.	Moderada; los cambios pueden realizarse entre sprints.	Alta; los cambios se incorporan en cada iteración.
Entrega de valor	Continua; entrega a medida que se completan las tareas.	Incremental; al final de cada sprint.	Incremental; al final de cada iteración con énfasis en calidad técnica.
Uso ideal	Equipos con flujos de trabajo constantes o no predecibles.	Proyectos con entregas regulares y necesidad de roles definidos.	Proyectos de software con alta necesidad de calidad técnica y cambios frecuentes.

Para seleccionar una metodología adecuada, es fundamental considerar las características del proyecto, los objetivos específicos, la naturaleza del equipo de trabajo y la adaptabilidad a cambios en los requisitos. Factores como la complejidad técnica, la necesidad de entregas incrementales y la interacción con el cliente también influyen en esta decisión. Tras evaluar estas variables, se decide utilizar la metodología de Programación Extrema (XP) debido a que esta es una metodología que promueve la comunicación constante y

fluida con el cliente, y al ser una programadora única, se fomenta la colaboración continua con un miembro del taller durante el proceso de desarrollo. La estructura iterativa y flexible de XP se adapta perfectamente a esta dinámica, permitiendo realizar entregas frecuentes y ajustes rápidos basados en las necesidades cambiantes del cliente. En conjunto, estas características la hacen una opción ideal para asegurar que el sistema se ajuste a las expectativas del taller en tiempo real. A continuación, se detalla dicha metodología.

1.3.1. Programación Extrema (*Extreme Programming* o XP)

La Programación Extrema o XP es una disciplina del negocio del desarrollo de software que centra a todo el equipo en objetivos comunes y alcanzables (Beck y Andres, 2004). Desarrollada por Kent Beck[3] en la década de 1990, XP se basa en un conjunto de prácticas y valores que promueven la colaboración constante entre los desarrolladores y los clientes, así como la entrega frecuente de versiones funcionales del software. Beck define un conjunto de cinco valores que establecen el fundamento para todo trabajo realizado como parte de XP: comunicación, simplicidad, retroalimentación, valentía y respeto. Cada uno de estos valores se usa como un motor para actividades, acciones y tareas específicas de XP (Pressman, 2010).

- Comunicación: la comunicación constante entre los miembros del equipo y con los clientes es esencial en XP. Se fomenta el diálogo abierto para asegurar que todos los involucrados comprendan los requisitos y objetivos del proyecto, lo que ayuda a prevenir malentendidos y errores en el desarrollo.

- Simplicidad: la simplicidad es el valor intelectual más intenso de XP. La idea es que el equipo debe enfocarse en resolver solo lo que es necesario en el momento, evitando la sobre complicación del código y las funcionalidades. Esto facilita el mantenimiento y la evolución del software a lo largo del tiempo.

- Retroalimentación: XP enfatiza la importancia de recibir retroalimentación continua a través de pruebas frecuentes y revisiones del código. Esto permite a los desarrolladores identificar problemas rápidamente y realizar ajustes antes de que se conviertan en fallos mayores. La retroalimentación también se obtiene del cliente, lo que asegura que el producto final cumpla con sus expectativas.

- Coraje: el coraje es una acción eficaz frente al miedo. Este se refiere a la disposición del equipo para tomar decisiones difíciles, como refactorizar el código o cambiar requisitos cuando sea necesario. El coraje también implica enfrentar problemas directamente y no posponer su resolución.

- Respeto: en un entorno de XP, se fomenta el respeto mutuo entre todos los miembros del equipo. Cada persona aporta habilidades y conocimientos únicos, y es fundamental valorar las contribuciones de los

[3]Kent Beck, nacido el 31 de marzo de 1961, es un ingeniero de software estadounidense y el creador de la Programación Extrema, una metodología de desarrollo de software que evita las especificaciones formales rígidas en favor de un proceso de diseño colaborativo e iterativo. Beck fue uno de los 17 firmantes originales del Manifiesto Ágil (Trifork, 2024).

demás para crear un ambiente colaborativo y productivo (Beck y Andres, 2004).

La programación extrema usa un enfoque orientado a objetos como paradigma preferido de desarrollo, y engloba un conjunto de reglas y prácticas que ocurren en el contexto de cuatro actividades estructurales: planeación, diseño, codificación y pruebas (Pressman, 2010).

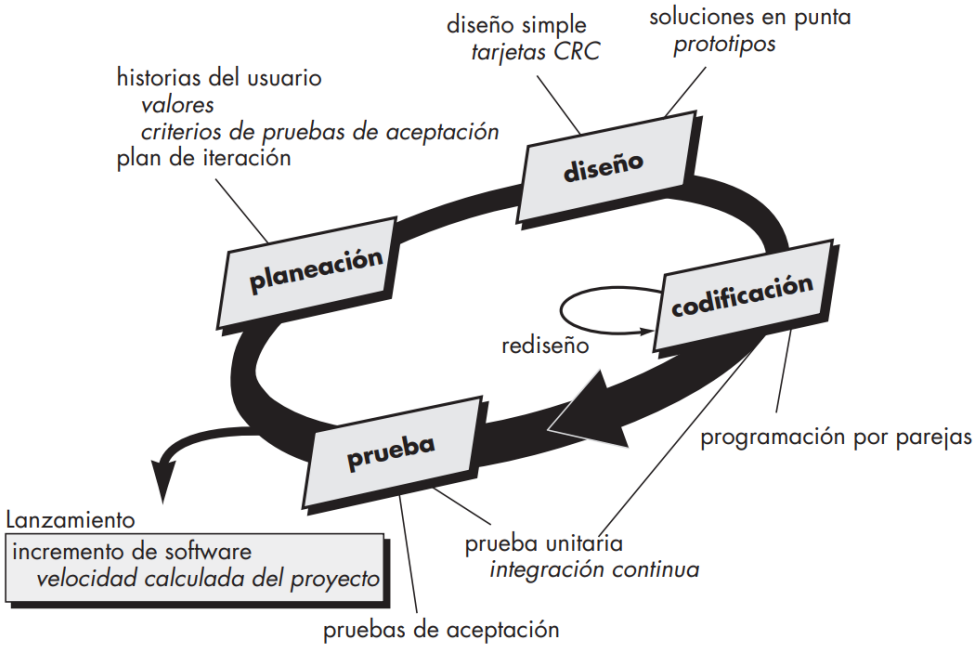


Figura 1.3. El proceso de la Programación Extrema [Tomado de: (Pressman, 2010).

1.4. Herramienta CASE (*Computer Aided Software Engineering*)

La ingeniería de software asistida por computadora (CASE) describe un amplio conjunto de herramientas y métodos, que ahorran mano de obra, y se utilizan en el desarrollo de software y el modelado de procesos comerciales. Las herramientas CASE crean un marco estandarizado y reutilizable para administrar proyectos de software y están destinadas a ayudar a los usuarios a mejorar la productividad y mantenerse organizados durante todo el proyecto (Awati, 2017).

1.4.1. Visual Paradigm v8.0

Visual Paradigm es una herramienta CASE (Computer Aided Software Engineering) ampliamente reconocida en el ámbito del desarrollo de software. Esta plataforma proporciona un entorno integral para el diseño, modelado y documentación de sistemas, lo que la convierte en una opción ideal para desarrolladores y equipos de trabajo que buscan mejorar su productividad y calidad en el proceso de desarrollo. Visual Paradigm

permite la creación de diagramas UML (*Unified Modeling Language*)[4] , así como otros tipos de diagramas que proporcionan una visualización y comprensión de los requisitos del sistema (Paradigm, 2024).

Este se destaca por su capacidad de integración con herramientas de gestión de proyectos y control de versiones, que permiten un seguimiento efectivo del progreso y cambios realizados en el desarrollo del software. Otra ventaja significativa, es su capacidad para generar automáticamente código a partir de los modelos diseñados. Esto no solo acelera el proceso de desarrollo, sino que también reduce la posibilidad de errores al transformar los modelos conceptuales en implementaciones funcionales. Soporta múltiples lenguajes de programación, lo que la hace versátil para diferentes tipos de proyectos (ibíd.).

A partir de los elementos antes expuestos, se decide utilizar Visual Paradigm en su versión 8.0 como herramienta CASE debido a que permite la creación de múltiples tipos de diagramas, lo que es clave para abordar la complejidad de la propuesta. Su capacidad para generar código directamente desde los diagramas permite reducir el tiempo necesario para la implementación de la solución, lo que asegura una mayor consistencia entre el diseño y la codificación. Se tuvo en cuenta, además, que la Universidad de las Ciencias Informáticas posee una licencia académica para su uso.

1.5. Lenguaje de modelado

Un lenguaje de modelado es un conjunto de notaciones y reglas que permiten la representación visual de los componentes, relaciones y comportamientos de un sistema. Estos lenguajes son esenciales en el desarrollo de software, ya que facilitan la comprensión, diseño y documentación de sistemas complejos. Al proporcionar una forma estandarizada de visualizar diferentes aspectos del software, los lenguajes de modelado ayudan a los equipos a comunicar ideas y a asegurar que todos los miembros comprendan los requisitos y la arquitectura del sistema (Fowler y Scott, 1999).

1.5.1. Lenguaje de Modelado Unificado (UML) v2.0

El UML (*Unified Modeling Language* o Lenguaje de Modelado Unificado) es un lenguaje de modelado visual, indispensable para la arquitectura y la ingeniería de software y sistemas. Este lenguaje fue pensado y creado como una lengua franca o lengua universal para los desarrolladores, o, en otras palabras: un lenguaje para simplificar y unificar lenguajes de modelación (Brutti, 2023). Algunas de sus características principales son:

[4]Un diagrama UML es un diagrama basado en el lenguaje UML (Unified Modeling Language) con el propósito de representar visualmente un sistema junto con sus principales actores, roles, acciones, artefactos o clases, con el fin de comprender, alterar, mantener o documentar mejor la información sobre el sistema (Kothari, 2024).

- Dinamismo: como lenguaje basado en notaciones y diagramas, es perfecto para el desarrollo y la simplificación de todo tipo de sistemas de software, ya sean simples o complejos.

- Claridad: la notación de este lenguaje, simple y unificada, es muy eficaz para los desarrolladores. Por tanto, no queda espacio para la ambigüedad, la vaguedad de conceptos, ni la confusión. Y por ello, resulta más fácil de entender y más útil al momento de coordinarse con otros equipos.

- Entendimiento: este modelo se basa en diagramas comprensibles para el usuario común, similares a esquemas o mapas conceptuales.

- Estandarización: UML es el lenguaje de modelaje de código más común en la actualidad. Por tanto, cuenta con una amplia gama de recursos para manejarlo, así como una gran comunidad de desarrolladores.

- Versatilidad: este modelo no se limita al modelado visual de software, sino a la creación e implementación de metodología ágiles, así como a la planificación y análisis de riesgos al implementar sistemas, entre otros casos de uso (Brutti, 2023).

La elección de utilizar UML en su versión 2.0 se sustenta en su capacidad para representar de manera clara y precisa tanto la estructura como el comportamiento del sistema que se pretende desarrollar. Dado que UML es un estándar ampliamente aceptado en la industria y estimula comunicación efectiva entre los diferentes actores involucrados en el proceso de desarrollo, resulta una herramienta ideal para la solución.

1.6. Herramienta de prototipado

El prototipado es una etapa crucial en el desarrollo de productos y proyectos, ya que permite visualizar y probar ideas antes de invertir recursos en su implementación. Las herramientas de prototipado digital son ampliamente utilizadas en la actualidad debido a su flexibilidad y facilidad de uso. Estas permiten crear prototipos interactivos que simulan la experiencia del producto final (IdeaFoster, 2023).

1.6.1. Canva v2.5

Canva es una plataforma digital orientada al diseño gráfico que se caracteriza por su accesibilidad y versatilidad. Fue desarrollada con el propósito de simplificar la creación de contenidos visuales, permitiendo a los usuarios generar piezas gráficas a través de una interfaz intuitiva. Esta herramienta proporciona acceso a una amplia gama de plantillas predefinidas y elementos gráficos, lo que reduce significativamente las barreras técnicas para el diseño. Entre sus aplicaciones más destacadas se incluyen la elaboración de presentaciones, publicaciones para redes sociales, logotipos, documentos y materiales publicitarios (Moro Ramos, 2024).

En términos funcionales, Canva combina un modelo *freemium*, que ofrece herramientas básicas de acceso gratuito, con opciones avanzadas disponibles en su versión Pro, las cuales incluyen almacenamiento ampliado y recursos gráficos exclusivos. Su compatibilidad con dispositivos móviles y de escritorio, junto con la integración de funciones colaborativas, la posicionan como una herramienta eficiente tanto para usuarios individuales como para equipos de trabajo (Moro Ramos, 2024).

Se decide utilizar Canva como herramienta de prototipado debido a su capacidad para facilitar la creación rápida de representaciones visuales de conceptos e ideas. Su interfaz amigable permite realizar diseños de manera eficiente, mientras que sus recursos gráficos predefinidos, agilizan el desarrollo de prototipos con apariencia profesional sin requerir habilidades avanzadas de diseño.

1.7. Lenguajes de programación

Un lenguaje de programación es un conjunto de reglas y sintaxis que permiten a los desarrolladores comunicarse con las computadoras, para escribir instrucciones y algoritmos que estas pueden ejecutar. Estos lenguajes son esenciales para la creación de software, ya que permiten describir procesos lógicos y controlar el comportamiento de los sistemas informáticos (Sebesta, 2015). En esta sección se detallan los lenguajes de programación seleccionados para llevar a cabo el desarrollo de la propuesta.

1.7.1. TypeScript v5.6

TypeScript es un lenguaje de programación desarrollado por Microsoft[5], que se presenta como un superconjunto de JavaScript[6]. Esto significa que todo el código JavaScript también es válido en TypeScript, lo que permite a los desarrolladores aprovechar las características de JavaScript mientras añaden funcionalidades adicionales. El uso de este lenguaje ha crecido significativamente, especialmente con el auge de *frameworks* modernos como Angular, que está completamente construido en TypeScript. Esto ha llevado a una mayor adopción del mismo por parte de los desarrolladores, que buscan crear aplicaciones robustas y escalables. (Ramirez, 2024). Dentro de sus carcacterísticas figuran:

- Tipado estático: TypeScript introduce la seguridad de tipos, lo que significa que permite a los desarrolladores especificar el tipo de datos que una variable puede contener. Esto ayuda a detectar errores durante la compilación en lugar de en tiempo de ejecución, lo que resulta en aplicaciones más robustas y menos propensas a fallos.

[5]Microsoft Corporation es una multinacional tecnológica de origen estadounidense que desarrolla y comercializa una amplia gama de software, hardware y servicios de tecnología de la información(TI), tanto para empresas como para consumidores privados (Rautenstrauch, 2024)

[6]JavaScript es un lenguaje de programación de código abierto diseñado para crear aplicaciones web (Sufiyan, 2024).

- Compatibilidad con JavaScript: es compatible con JavaScript, lo que permite a los desarrolladores integrar su código de JavaScript existente en proyectos de TypeScript, sin necesidad de reescribirlo.

- Soporte para Programación Orientada a Objetos (POO): este lenguaje extiende las capacidades de JavaScript en términos de programación orientada a objetos, permitiendo el uso de clases, interfaces y herencia. Esto ayuda a organizar el código y aplicar principios como DRY *(Don't Repeat Yourself)*, lo que mejora la mantenibilidad del software.

- Herramientas y ecosistema: cuenta con un amplio ecosistema de herramientas para desarrolladores, incluyendo editores que ofrecen autocompletado, documentación en línea y resaltado de errores, lo cual optimiza el proceso de depuración.

- Usabilidad en diferentes entornos: TypeScript no solo se limita al desarrollo *frontend*[7], también se utiliza para construir servicios *backend*[8] y aplicaciones de línea de comandos, lo que amplía su aplicabilidad en diversos contextos de desarrollo (Goldberg, 2022).

Se decide utilizar TypeScript en su versión 5.6 principalmente por su capacidad para mejorar la calidad del código y facilitar la escalabilidad en proyectos complejos. Al ser una extensión de JavaScript con tipado estático opcional, permite detectar errores durante el desarrollo, lo que contribuye a un proceso de depuración más seguro y a un código más robusto.

1.7.2. Python v3.13.0

Python es un lenguaje de programación de alto nivel, interpretado y de código abierto, creado por Guido van Rossum[9] en 1991. Se centra en la simplicidad y la legibilidad del código, hecho que permite a los desarrolladores expresar conceptos en menos líneas de código en comparación con otros lenguajes como C++ o Java. Además de por su simplicidad, Python es reconocido por ofrecer funcionalidades que lo hacen destacar por delante de la gran mayoría de los lenguajes de programación (Toro Bonilla, 2022). Algunas de sus características más notables son:

- Sencillez y legibilidad: Python está diseñado para ser fácil de aprender y utilizar, con una sintaxis que se asemeja al lenguaje humano. Esto permite a los desarrolladores escribir un código intuitivo, facilitando su comprensión y mantenimiento.

[7] El *frontend* es la parte de un programa, sitio web o dispositivo a la que un usuario puede acceder directamente. Es la cara del sitio, ubicada en el lado del cliente (Chapaval, 2017).

[8] El *backend* se encarga de la conexión con la base de datos y el servidor utilizado por el sitio web. Esta parte está situada en el lado del servidor (ibíd.).

[9] Guido van Rossum nacido el 31 de enero de 1956, es un programador informático holandés, ampliamente reconocido como el cerebro detrás de la creación del lenguaje de programación Python (Matczak, 2023).

- Amplia variedad de bibliotecas y *frameworks*: el lenguaje cuenta con una extensa colección de bibliotecas y *frameworks* que facilitan el desarrollo en diversas áreas, como análisis de datos, inteligencia artificial y desarrollo web. Esto permite a los programadores aprovechar soluciones ya existentes para acelerar su trabajo.

- Versatilidad: se utiliza en una amplia gama de aplicaciones, desde desarrollo web hasta análisis de *big data*[10] e inteligencia artificial. Su flexibilidad lo convierte en una opción popular para muchos tipos de proyectos.

- Soporte para Programación Orientada a Objetos (POO): permite la programación orientada a objetos, lo que facilita la creación de aplicaciones complejas mediante la encapsulación y reutilización del código (Cristancho, 2022).

Estas características de Python convierten a este lenguaje de programación en una herramienta fiable y flexible para desarrolladores de todos los niveles. La elección del mismo en su versión 3.13.0 se sustenta en las características ya mencionadas. Otra razón clave, es que la autora tiene conocimiento de este lenguaje, lo que favorece un desarrollo más sencillo y directo del sistema.

1.8. Lenguaje de Marcas de Hipertexto HTML5

HTML (Lenguaje de Marcas de Hipertexto, del inglés *HyperText Markup Language*) es el lenguaje de marcado estándar utilizado para crear y estructurar contenido en la web. Este lenguaje permite a los desarrolladores definir la estructura de las páginas web mediante el uso de etiquetas que indican cómo se debe mostrar el contenido, como texto, imágenes y enlaces (Gauchat, 2012).

Su código fuente, junto con el CSS *(Cascading Style Sheets)*, muestra en los navegadores cómo se compone visualmente la página web en el dispositivo final, incluyendo el diseño, la tipografía y los colores. Con la versión actual HTML5 están disponibles nuevas posibilidades de uso y atributos HTML. Entre sus usos y funciones se incluyen:

- Estructura jerárquica: HTML utiliza una estructura de etiquetas anidadas que permite organizar el contenido de manera lógica.

- Semántica: las etiquetas tienen significados específicos que ayudan a describir el tipo de contenido, lo que mejora la accesibilidad y la optimización para motores de búsqueda.

[10]El término *big data* hace referencia a conjuntos de datos extremadamente grandes y complejos que no se pueden gestionar ni analizar fácilmente con herramientas de procesamiento de datos tradicionales, en particular hojas de cálculo (Chen, 2024).

- Multimedia: permite la integración de diferentes tipos de contenido multimedia, como imágenes, videos y audio.

- Enlaces: facilita la creación de enlaces entre diferentes páginas web, promoviendo la navegación.

- Compatibilidad: es compatible con todos los navegadores modernos, lo que asegura que las páginas se muestren correctamente en diversas plataformas.

- Interactividad: aunque HTML por sí solo no proporciona interactividad, se puede combinar fácilmente con CSS y JavaScript para crear experiencias dinámicas (Gauchat, 2012).

La elección de HTML 5 se justifica por su papel fundamental en la creación de la estructura del sitio web para el taller Fotoché. Al ser el lenguaje base para el desarrollo web, HTML permite organizar y presentar la información de una forma clara y accesible. Su compatibilidad con otros lenguajes y tecnologías asegura que el sistema pueda evolucionar fácilmente en el futuro, integrando nuevas funcionalidades según sea necesario. Además, su uso generalizado garantiza que cualquier desarrollador pueda trabajar con el código sin dificultades, lo que facilita el mantenimiento y las actualizaciones del sistema.

1.9. Lenguaje de Hojas de Estilo en Cascada

CSS, acrónimo de *Cascading Style Sheets* (Hojas de Estilo en Cascada), es un lenguaje de diseño utilizado en el desarrollo web para controlar la presentación de documentos HTML. Su propósito principal es separar el contenido estructural de una página web de su diseño visual, lo que permite mayor flexibilidad y mantenimiento del código (Schulz, 2009). Entre las características de CSS se incluyen:

- Flexibilidad: CSS permite crear diseños adaptables que se ajustan a diferentes tamaños de pantalla, desde dispositivos móviles hasta computadoras de escritorio, que garantizan una experiencia óptima en cualquier dispositivo.

- Reutilización: fomenta la reutilización del código al definir estilos una vez y aplicarlos a múltiples elementos o páginas. Esto asegura consistencia en el diseño y simplifica el mantenimiento de los proyectos web.

- Separación de contenido y presentación: se centra en mantener separados el contenido estructural (HTML) y el diseño visual. Esto facilita la modificación del aspecto del sitio sin alterar el contenido, optimizando actualizaciones y reduciendo errores.

- Cascada y especificidad: la funcionalidad en cascada asegura que, cuando múltiples reglas afectan un mismo elemento, se apliquen según un orden jerárquico basado en la especificidad y la precedencia. Este concepto permite un control detallado del diseño.

- Soporte para animaciones y efectos: CSS3 introdujo herramientas avanzadas para animaciones y efectos visuales. transiciones y transformaciones permiten incorporar interactividad y dinamismo al diseño de sitios web de manera eficiente y atractiva (Schulz, 2009).

Se decide utilizar CSS en su versión 3 como lenguaje de estilo debido a su capacidad para mejorar la experiencia del usuario a través de un diseño visual atractivo y funcional. Esto es fundamental para atraer y retener clientes, ya que un diseño bien estructurado contribuye significativamente a la satisfacción del usuario. Además, la flexibilidad que ofrece CSS permite realizar ajustes rápidos en el diseño según las necesidades del negocio, facilitando así la adaptación a cambios futuros.

1.10. Bibliotecas y marcos de trabajo para el desarrollo

El *framework* es un término que proviene del inglés y significa marco de trabajo o estructura. En el ámbito de la programación, un *framework* es un conjunto de reglas y convenciones que se usan para desarrollar software de manera rápida. Estos marcos de trabajo se emplean para ahorrar tiempo y esfuerzo en el desarrollo de aplicaciones, ya que proporcionan una estructura básica que se puede utilizar como punto de partida. Además, ofrecen soluciones a problemas comunes en el desarrollo de software, lo que significa que los desarrolladores pueden centrarse en las funcionalidades específicas de su aplicación en lugar de perder tiempo resolviendo problemas técnicos (Lucena, 2023).

1.10.1. React v19

React es una biblioteca de JavaScript que se utiliza para construir interfaces de usuario interactivas. Permite a los desarrolladores crear aplicaciones dinámicas mediante la creación de componentes reutilizables. Esta capacidad de modularidad es una de sus características más destacadas, ya que posibilita dividir la interfaz en partes independientes que pueden gestionarse y actualizarse de forma competente (Gackenheimer, 2015). Algunas de sus características clave son:

- Componentes: organiza la interfaz de usuario en componentes reutilizables, que pueden ser de distinto tipo y tamaño. De esta forma, simplifica la construcción y el mantenimiento de las aplicaciones, ya que cada componente se puede desarrollar y probar de forma aislada antes de integrarlo en el desarrollo final.

- Estados y desarrollo declarativo: permite describir cómo debe verse la interfaz dependiendo del estado de la aplicación. De este modo, cuando un componente cambia de estado, React hace un seguimiento

de esos cambios y actualiza la interfaz de usuario para que aparezca acorde al nuevo estado. Esto, además de simplificar la gestión del estado, implica que la interfaz siempre estará sincronizada y actualizada con los datos de la aplicación.

- Separación de contenido y presentación: se centra en mantener separados el contenido estructural (HTML) y el diseño visual. Esto facilita la modificación del aspecto del sitio sin alterar el contenido, optimizando actualizaciones y reduciendo errores.

- Virtual DOM[11]: React aplica las modificaciones primero en una representación virtual del DOM, con el objetivo de que posteriormente solo se apliquen al DOM real los cambios necesarios, reduciendo la carga en el navegador.

- Unidireccionalidad de datos: los datos solo fluyen en una dirección a través de la aplicación. Esto facilita el seguimiento y el mantenimiento, con lo que se consigue un código más predecible y fácil de depurar (Gackenheimer, 2015).

En cuanto a su implementación, React es compatible con una gran cantidad de tecnologías y plataformas, lo que le permite ser utilizado en aplicaciones multiplataforma, desde sitios web hasta aplicaciones móviles. Se decide utilizar React en su versión 19 por su enfoque modular, que facilita el desarrollo de aplicaciones escalables y de fácil mantenimiento. Además, permite una integración fluida con otras bibliotecas o *frameworks*, lo que brinda flexibilidad para adaptarse a las necesidades específicas del proyecto.

1.10.2. Django v5.0

Django es un *framework* de desarrollo web de alto nivel que permite a los desarrolladores construir aplicaciones en un menor tiempo. Está basado en Python y sigue el patrón de diseño Modelo-Vista-Plantilla (MTV), lo que facilita la organización del código y su mantenimiento a largo plazo (Rubio, 2017). Algunas de las características de Django son:

- Desarrollo rápido: Django está diseñado para ayudar a los desarrolladores a tomar menos decisiones sobre la configuración y a centrarse en la construcción de aplicaciones. Su objetivo es reducir el tiempo de desarrollo.

- Escalabilidad: maneja aplicaciones desde pequeñas hasta grandes, lo que permite un crecimiento fácil sin comprometer el rendimiento.

[11]El DOM (*Document Object Model* o Modelo de Objetos de Documento) es una interfaz que permite que un lenguaje de programación manipule el contenido, la estructura y el estilo de un sitio web. Se suele denominar árbol DOM y consiste en un árbol de objetos llamados nodos (Rascia, 2023).

- Administración automática: incluye una interfaz de administración automática, posibilitando la gestión de datos en el *backend* sin necesidad de construir paneles personalizados.

- Sistema de plantillas: este *framework* utiliza un sistema de plantillas que permite separar la lógica de la presentación, facilitando así el trabajo entre diseñadores y desarrolladores (D. S. Foundation, 2024).

La elección de utilizar Django en su versión 5.0 se basa en su capacidad para acelerar el desarrollo mediante su enfoque en ofrecer funcionalidades listas para usar, como la administración automática y su ORM[12]. Proporciona un alto nivel de seguridad, lo cual es crucial para proteger los datos sensibles de los clientes y las operaciones del sistema. Al estar basado en Python, lenguaje que la autora domina, facilita la integración y el manejo de la lógica del negocio, permitiendo un desarrollo mantenible.

1.11. Entorno de desarrollo

Un entorno de desarrollo integrado (IDE) es un espacio de trabajo que permite a los desarrolladores crear una aplicación o realizar cambios en ella sin afectar a la versión real del producto de software. Estos cambios pueden incluir el mantenimiento, la depuración y la aplicación de parches (Vargas, 2022).

1.11.1. Visual Studio Code v1.93.1

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft, diseñado para ser ligero, rápido y altamente personalizable. A diferencia de otros editores de código, como los IDE completos (Entornos de Desarrollo Integrados), este se enfoca en ofrecer una experiencia ágil para la escritura y edición de código, sin sacrificar características avanzadas (Cimas Cuadrado, 2024).

Al ser una herramienta multiplataforma y gratuita, facilita su adopción tanto por parte de desarrolladores principiantes como por equipos profesionales que buscan una herramienta rentable sin tener que invertir en costosas licencias de software. Entre sus características clave, se incluyen la autocompletado inteligente mediante *IntelliSense*[13], la depuración integrada, y el control de versiones con Git[14], lo que convierte a VS Code en una opción versátil tanto para desarrolladores individuales como para equipos de desarrollo (Cimas Cuadrado, 2024).

[12] El ORM (*Object Relational Mapping* o Mapeo Relacional de Objetos) es una técnica utilizada para crear un puente entre los programas orientados a objetos y, en la mayoría de los casos, las bases de datos relacionales. Se puede ver el ORM como la capa que conecta la programación orientada a objetos (OOP) con las bases de datos relacionales (Abba, 2022).

[13] *IntelliSense* es una herramienta de completado de código integrada en Visual Studio. Forma parte de una serie de herramientas similares que permiten completar código o texto de forma inteligente en distintas plataformas (Rouse, 2024).

[14] Git es un sistema de control de versiones distribuido , gratuito y de código abierto, diseñado para gestionar todo, desde proyectos pequeños hasta muy grandes, con velocidad y eficiencia (Straub y Chacon, 2014).

Se decide utilizar Visual Studio Code en su versión 1.93.1 debido a su flexibilidad, rendimiento ligero y capacidad de integración con diversas herramientas y tecnologías de desarrollo. Su soporte para múltiples lenguajes de programación y su amplia biblioteca de extensiones lo convierten en una opción ideal para adaptarse a las necesidades específicas del proyecto.

1.12. Sistema gestor de base de datos

Un sistema gestor de base de datos (SGBD) o *Database Management System* (DBMS) es un conjunto de programas invisibles para el usuario final con el que se administra y gestiona la información que incluye una base de datos. Entre sus funciones se encuentran la de permitir a los usuarios de negocio almacenar la información, modificar datos y acceder a los activos de conocimiento de la organización. Asimismo, el gestor de base de datos también se ocupa de realizar consultas y hacer análisis para generar informes (Darias Perez, 2021).

1.12.1. PostgreSQL v16.0

PostgreSQL es un sistema de bases de datos de código abierto que se destaca por su capacidad de adaptación y su cumplimiento con el estándar SQL. Este sistema ofrece flexibilidad y extensibilidad, permitiendo a los usuarios adaptar la base de datos a sus necesidades específicas mediante la creación de nuevas funciones y tipos de datos (Dorantes, 2015). Algunas de las características principales de PostgreSQL son:

- Transacciones ACID: garantiza transacciones con propiedades de Atomicidad, Consistencia, Aislamiento y Durabilidad, lo que asegura la integridad de los datos en situaciones complejas.

- Soporte para replicación: permite la replicación de datos entre servidores, lo cual es esencial para garantizar la disponibilidad y redundancia en aplicaciones críticas.

- Extensibilidad: los usuarios pueden agregar nuevos tipos de datos, funciones, operadores, y lenguajes procedurales, lo que lo hace altamente personalizable.

- Soporte para JSON[15] y XML[16]: además de datos estructurados, permite almacenar y consultar datos no estructurados, lo que lo convierte en una opción versátil para aplicaciones modernas (Group, 2024).

Se selecciona PostgreSQL v16.0 debido a su robustez y flexibilidad para manejar tanto grandes volúmenes de datos como necesidades específicas de personalización. Su naturaleza de código abierto permite la inte-

[15]JSON *(JavaScript Object Notation)* es un formato ligero de intercambio de datos. Se utiliza para representar estructuras de datos compuestas por pares clave-valor (Tagliaferri, 2022).

[16]XML *(eXtensible Markup Language)* es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos (Souza, 2019).

gración con otros sistemas y tecnologías modernas, garantizando una solución adaptable a las demandas del taller.

1.13. Control de versiones

El control de versiones, también conocido como control de código fuente, es la práctica de realizar un seguimiento y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo (Atlassian, 2024).

1.13.1. Git v2.2

Git es un sistema de control de versiones distribuido que permite a los desarrolladores gestionar y seguir los cambios realizados en su código a lo largo del tiempo. Este sistema facilita la colaboración entre múltiples usuarios al permitir que trabajen en proyectos de forma simultánea sin interferir en el trabajo de los demás. La naturaleza distribuida de Git permite que cada colaborador tenga la historia completa del proyecto en su máquina local, lo que facilita el trabajo sin necesidad de conexión constante a un servidor central. Esto no solo mejora la independencia, sino que también contribuye a una mayor seguridad del código, ya que se reduce el riesgo de pérdida de datos en un solo punto de fallo (Straub y Chacon, 2014).

La decisión de utilizar Git versión 2.2 se basa en su capacidad para gestionar eficazmente los cambios en el código, incluso cuando en el presente proyecto solo hay un desarrollador. Git permite llevar un registro detallado de las modificaciones realizadas, facilita la organización del trabajo y ofrece la posibilidad de revertir a versiones anteriores si es necesario. Además, su naturaleza distribuida proporciona una copia completa del historial del proyecto, lo que asegura la integridad del código y reduce el riesgo de pérdida de datos.

1.14. Conclusiones parciales

En este capítulo se examinaron los conceptos teóricos que respaldan la propuesta de solución a la problemática identificada, lo que permitió arribar a las siguientes conclusiones:

- El análisis de los conceptos clave relacionados con la gestión de los recursos y servicios en talleres de impresión, sentó las bases teóricas necesarias para comprender el problema planteado.

- El estudio de los sistemas homólogos evidenció la necesidad de desarrollar una solución personalizada para automatizar las operaciones del taller de impresiones Fotoché.

- La elección de la metodología XP se ajustó al ciclo de vida del proyecto, con el objetivo de garantizar un desarrollo ágil y flexible.

- La decisión de utilizar tecnologías modernas y libres, aseguró la escalabilidad de la solución propuesta para la soberanía tecnológica a la que aspira el país.

CAPÍTULO 2

PLANEACIÓN Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

En este capítulo se presentan los elementos clave relacionados con la propuesta de solución. Se generan los artefactos correspondientes a la metodología seleccionada y se identifican los requisitos funcionales y no funcionales que debe cumplir el sistema, encapsulando los primeros en historias de usuario. De igual forma se fundamentan la arquitectura de software y los patrones de diseño empleados, además se muestran las tarjetas CRC(Clase-Responsabilidad-Colaborador) como un mecanismo eficaz para pensar en el software en un contexto orientado a objetos.

2.1. Descripción de la propuesta de solución

La propuesta de solución se centra en el desarrollo de un sistema de gestión web para el taller de impresiones Fotoché, orientado a automatizar las tareas relacionadas con la gestión de los recursos y servicios, permitiendo llevar un control en las operaciones del taller. A través de esta herramienta, se busca mejorar la gestión operativa, facilitando el seguimiento de existencias y la planificación de aprovisionamientos.

El sistema estará compuesto por dos módulos principales: el módulo administrador y el módulo trabajador. El módulo administrador estará diseñado para proporcionar a los gestores del taller un control integral de la gestión de los recursos y servicios. Entre sus funciones se incluyen el registro, modificación y eliminación de recursos y servicios, así como la visualización de gastos de recursos y la obtención de ganancias por la prestación de servicios. Este no solo permitirá llevar un control en tiempo real del uso de los recursos, sino que también facilitará el análisis detallado del rendimiento económico del taller, contribuyendo así a una gestión más informada y estratégica.

Por otro lado, el módulo trabajador se enfocará en registrar las operaciones diarias del personal del taller. Este componente permitirá a los empleados registrar cada servicio realizado, incluyendo detalles como el uso de recursos y el empleado responsable. De esta manera, se logrará un control centralizado sobre la productividad y el rendimiento del equipo. La interacción entre ambos módulos asegurará un mejor flujo de

información dentro del sistema.

2.1.1. Usuarios relacionados con el sistema

Con base en la descripción de la propuesta de solución, se identifican dos roles principales relacionados con el Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché: Rol Administrador y Rol Trabajador. Cada uno con responsabilidades específicas y niveles de acceso diferenciados. La clasificación en roles tiene como objetivo facilitar el flujo de trabajo y la precisión en el control de recursos y servicios, lo cual se logra mediante funcionalidades ajustadas a las necesidades de cada perfil.

Tabla 2.1. Usuarios relacionados con el sistema.

Rol	Descripción
Administrador	Encargado de gestionar el sistema en su totalidad, incluyendo el registro, modificación y eliminación de recursos y servicios. Tiene acceso a funciones avanzadas como la visualización de ganancias y gastos totales en fechas determinadas.
Trabajador	Encargado de registrar los servicios ofrecidos en el taller, incluyendo los recursos empleados, lo que permite mantener un control actualizado sobre la productividad diaria.

2.2. Disciplina de requisitos

Según (Sommerville, 2011) los requerimientos para un sistema son descripciones de lo que el sistema debe hacer: el servicio que ofrece y las restricciones en su operación. Estos se dividen generalmente en dos categorías: funcionales, que describen las funciones específicas del sistema, y no funcionales, que abarcan aspectos como el rendimiento, la seguridad y la usabilidad. La correcta identificación y documentación de estos requisitos es crucial para el éxito del desarrollo de software, ya que establecen las bases sobre las cuales se construye el sistema.

2.2.1. Requisitos funcionales (RF)

Los requisitos funcionales son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas (ibíd.). Estos requisitos son esenciales para guiar el desarrollo del software, ya que establecen claramente lo que se espera del sistema en términos de funcionalidades específicas.

Para identificar los requisitos funcionales correspondientes al Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché se realizó un levantamiento de información a través de una entrevista

con el cliente (Ver A.1), obteniendo un total de 29 requisitos funcionales enumerados y descritos en la siguiente tabla (Ver 2.2).

La asignación de complejidad y prioridad a estos requisitos se llevó a cabo mediante un análisis sistemático que integró aspectos técnicos y operativos del sistema. En términos técnicos, se realizó una evaluación de la complejidad mediante Puntos de Historia *(Story Points)*[1], asignando niveles de esfuerzo según el conocimiento de la autora en las tecnologías necesarias y la cantidad de tareas involucradas. Esta clasificación permitió organizar los requisitos en niveles de complejidad bajo, medio y alto.

Desde el punto de vista operativo, se estableció la prioridad de cada requisito considerando su relevancia en el negocio y su impacto en el flujo de trabajo de los usuarios. Factores como la frecuencia de uso y la urgencia expresada por el cliente fueron esenciales. De este modo, el análisis integró tanto la complejidad técnica como la prioridad operativa, proporcionando una clasificación adecuada para el desarrollo del sistema.

Tabla 2.2. Listado de requisitos funcionales del sistema.

No.	Requisito	Descripción	Prioridad	Complejidad
RF1	Autenticar usuario	El sistema debe permitir a los usuarios acceder a sus funcionalidades introduciendo los siguientes datos: • Usuario • Contraseña	Alta	Alta
RF2	Registrar usuario	El sistema debe permitir al administrador registrar un nuevo usuario compuesto por los siguientes datos: • Nombre y apellidos • Usuario • Contraseña • Rol	Alta	Alta

[1]Los *Story Points* o Puntos de Historia son estimaciones o proximidades de nivel superior que establecen los equipos de desarrollo y gestión de proyectos ágiles. Miden los niveles de dificultad para cumplir una determinada tarea o requisito empresarial (Kehr, 2024).

No.	Requisito	Descripción	Prioridad	Complejidad
RF3	Registrar recurso	El sistema debe permitir al administrador registrar un nuevo recurso compuesto por los siguientes datos: • Nombre • Cantidad • Costo	Alta	Alta
RF4	Modificar recurso	El sistema debe permitir al administrador modificar la información de un recurso ya existente: • Nombre • Cantidad • Costo	Alta	Media
RF5	Registrar servicio	El sistema debe permitir al administrador registrar un nuevo servicio mediante los siguientes datos: • Nombre • Descripción • Precio • Recursos que consume • Estado	Alta	Alta
RF6	Modificar servicio	El sistema debe permitir al administrador modificar un servicio ya existente: • Nombre • Descripción • Precio • Recursos que consume • Estado	Alta	Media
RF7	Ofrecer servicio	El sistema debe permitir a los empleados registrar la operación de un nuevo servicio. Al ofrecer un servicio, se almacenará del mismo los siguientes datos: • Tipo de servicio • Cantidad	Alta	Alta

No.	Requisito	Descripción	Prioridad	Complejidad
RF8	Visualizar ganancia total en una fecha determinada	El sistema debe permitir al administrador comprobar la ganancia total por los servicios ofrecidos en una fecha determinada.	Alta	Alta
RF9	Visualizar gasto total de recursos en una fecha determinada	El sistema debe permitir al administrador comprobar el gasto total de los recursos por los servicios ofrecidos en una fecha determinada.	Alta	Alta
RF10	Visualizar servicios ofrecidos en una fecha determinada	El sistema debe permitir al administrador visualizar los servicios ofrecidos por los trabajadores en una fecha determinada en forma de lista con los siguientes datos: • Fecha • Tipo de servicio • Trabajador que ofreció el servicio	Alta	Alta
RF11	Eliminar recurso	El sistema debe permitir al administrador eliminar un recurso existente.	Alta	Media
RF12	Listar recursos	El sistema debe permitir al administrador visualizar una lista de todos los recursos existentes. En el listado se deben mostrar los siguientes datos: • Nombre • Cantidad • Costo	Alta	Media
RF13	Buscar recursos	El sistema debe permitir al administrador buscar los recursos mediante los siguientes criterios de búsqueda: • Nombre	Alta	Media
RF14	Eliminar servicio	El sistema debe permitir al administrador eliminar un servicio existente.	Alta	Media
RF15	Listar servicios	El sistema debe permitir al administrador visualizar todos los servicios que se ofrecen en el negocio. En el listado se deben mostrar los siguientes datos: • Nombre • Precio • Estado	Alta	Media

No.	Requisito	Descripción	Prioridad	Complejidad
RF16	Buscar servicios	El sistema debe permitir al administrador buscar los servicios mediante los siguientes criterios de búsqueda: • Nombre	Alta	Media
RF17	Cerrar sesión	El sistema debe permitir a los usuarios cerrar la sesión activa.	Media	Media
RF18	Modificar usuario	El sistema debe permitir al administrador modificar la información de los usuarios ya registrados (el rol del usuario no puede ser modificado): • Nombre y apellidos • Usuario • Contraseña	Media	Media
RF19	Eliminar usuario	El sistema debe permitir al administrador eliminar un usuario.	Media	Media
RF20	Listar usuarios	El sistema debe permitir al administrador visualizar una lista de todos los usuarios. En el listado se deben mostrar los siguientes datos: • Nombre y apellidos • Usuario • Rol	Media	Media
RF21	Seleccionar todos los recursos	El sistema debe permitir al administrador seleccionar todos los recursos.	Baja	Baja
RF22	Deseleccionar todos los recursos	El sistema debe permitir al administrador deseleccionar todos los recursos que estén seleccionados.	Baja	Baja
RF23	Eliminar todos los recursos seleccionados	El sistema debe permitir al administrador eliminar todos los recursos que estén seleccionados.	Baja	Baja
RF24	Seleccionar todos los servicios	El sistema debe permitir al administrador seleccionar todos los servicios.	Baja	Baja
RF25	Deseleccionar todos los servicios	El sistema debe permitir al administrador deseleccionar todos los servicios que estén seleccionados.	Baja	Baja
RF26	Eliminar todos los servicios seleccionados	El sistema debe permitir al administrador eliminar todos los servicios que estén seleccionados.	Baja	Baja

No.	Requisito	Descripción	Prioridad	Complejidad
RF27	Seleccionar todos los usuarios	El sistema debe permitir al administrador seleccionar todos los usuarios.	Baja	Baja
RF28	Deseleccionar todos los usuarios	El sistema debe permitir al administrador deseleccionar todos los usuarios que estén seleccionados.	Baja	Baja
RF29	Eliminar todos los usuarios seleccionados	El sistema debe permitir al administrador eliminar todos los usuarios que estén seleccionados.	Baja	Baja

2.2.2. Requisitos no funcionales (RnF)

Los requisitos no funcionales son limitaciones sobre servicios o funciones que ofrece el sistema. Incluyen restricciones tanto de temporización y del proceso de desarrollo, como impuestas por los estándares (Sommerville, 2011). Estos requisitos son cruciales para garantizar que el software no solo cumpla con las funciones requeridas, sino que también ofrezca una experiencia de usuario satisfactoria y cumpla con estándares de calidad. A continuación, se listan dichos requisitos.

Interfaz o apariencia externa

RnF 1. El sistema debe ofrecer una interfaz con un diseño sencillo, con colores de gama como negro y amarillo claro.

RnF 2. El sistema debe garantizar una correcta organización de la información para permitir una adecuada interpretación.

Usabilidad

RnF 3. El sistema debe ser una aplicación web con una curva de aprendizaje baja, permitiendo que los usuarios se adapten rápidamente sin necesidad de capacitación extensa.

RnF 4. En el sistema se deben visualizar todos los mensajes en idioma español. La tipografía debe ser uniforme, de un tamaño adecuado.

Seguridad

RnF 5. Se deben establecer roles para acceder al sistema, garantizando que la información almacenada solo podrá ser modificada y/o visualizada por los usuarios autorizados.

RnF 6. Todos los datos sensibles deben ser encriptados tanto en tránsito como en reposo para garantizar la privacidad.

Mantenibilidad

RnF 7. Todo el código debe seguir estándares de codificación y estar debidamente documentado para facilitar futuras modificaciones y asegurar su escalabilidad a largo plazo.

Software

RnF 8. El sistema debe funcionar sobre cualquier distribución de sistema operativo Linux y/o superior a Windows 7.

RnF 9. El sistema debe ser accesible desde los principales navegadores web (Internet Explorer 9.0 o superior, Mozilla Firefox 5.0 o superior, Google Chrome 10.0 o superior, o cualquier otro).

Hardware

RnF 10. Las estaciones de trabajo deben contar con un procesador Intel Core i3 de 10ma generación o superior, RAM DDR4 de 8GB de o superior, Disco Duro (HDD) de 256GB como mínimo y puerto de red Ethernet LAN RJ-45.

RnF 11. El servidor debe contar con un Intel Core i5 de 10ma generación o superior, con una frecuencia de 2.1 GHz a 3.2 GHz, RAM DDR4 de 8GB o superior con una velocidad de DDR4 ECC (Error-Correcting Code) a 2666 MHz o superior, Disco Duro (HDD) de 256 GB como mínimo, soporte para Gigabit Ethernet (1 Gbps) y al menos 2 puertos Ethernet LAN RJ-45, uno de ellos destinado a redundancia o balanceo de carga.

RnF 12. Las estaciones de trabajo deben estar equipadas con una pantalla que posea una resolución mínima de 1920x1080 píxeles.

2.3. Fase I: Planeación

La actividad de planeación (también llamada juego de planeación) comienza escuchando, actividad para recabar requerimientos que permiten que los miembros técnicos del equipo XP entiendan el contexto del negocio para el software y adquieren la sensibilidad de la salida y características principales y funcionalidad que se requieren. Escuchar lleva a la creación de algunas "historias"(también llamadas historias del usuario) que describen la salida necesaria, características y funcionalidad del software que se va a elaborar. Cada historia (similar a los casos de usos) es escrita por el cliente y colocada en una tarjeta indizada. El cliente asigna un valor (es decir, una prioridad) a la historia con base en el valor general de la característica o función para el negocio. Después, los miembros del equipo XP evalúan cada historia y le asignan un costo, medido en semanas de desarrollo. Si se estima que la historia requiere más de 3 semanas de desarrollo, se pide al cliente que la descomponga en historias más chicas y de nuevo se asigna un valor y costo. Es importante observar que en cualquier momento es posible escribir nuevas historias (Pressman, 2010).

2.3.1. Historias de Usuario (HU)

Las historias de usuario tienen el mismo propósito que los casos de uso, pero no son lo mismo. Se utilizan para crear estimaciones de tiempo y en lugar de un gran documento de requerimientos. Estas son escritas por los clientes como cosas que el sistema tiene que hacer por ellos (Wells, 1999). En la presente investigación se definieron 29 historias de usuario, una por cada requisito funcional, de la cuales se presenta, las

correspondiente al RF 1: Autenticar usuario y al requisito funcional RF 2: Registrar usuario. Las restantes pueden ser consultadas en los anexos de la investigación (Ver A.3).

Tabla 2.3. Historia de usuario # 1

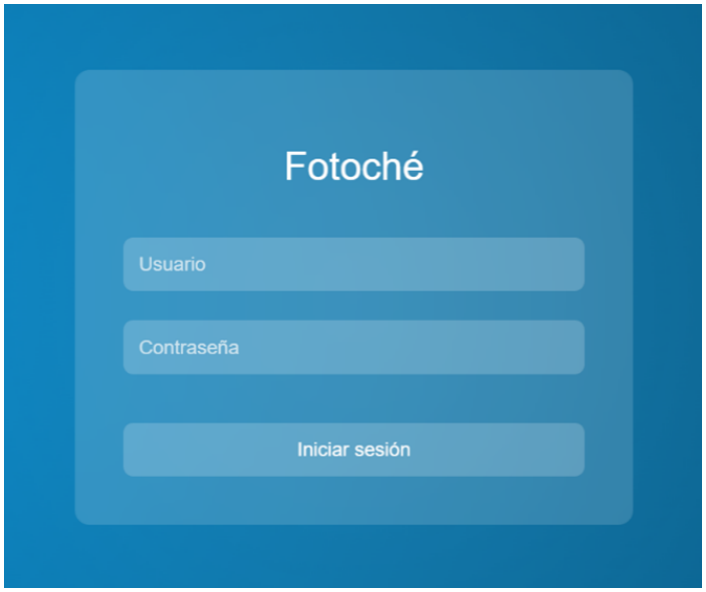
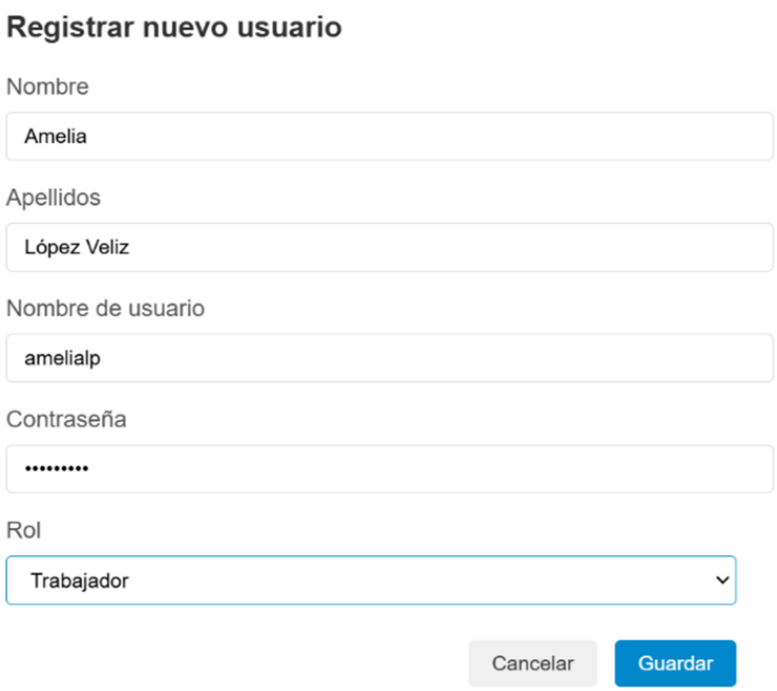
Historia de usuario	
Número: 1	**Nombre:** Autenticar usuario
Usuario: Administrador/Trabajador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá a los usuarios autenticarse en el sistema mediante un formulario de inicio de sesión llenando los campos Usuario y Contraseña.	
Observaciones: El sistema deberá proporcionar un mensaje de error en caso de que el usuario o la contraseña sean incorrectos. Los campos usuario y contraseña son obligatorios para acceder a cualquier funcionalidad del sistema.	
Interfaz: 	

Tabla 2.4. Historia de usuario # 2

Historia de usuario	
Número: 2	**Nombre:** Registrar usuario
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	

Continúa en la próxima página

Tabla 2.4. Continuación de la página anterior

Descripción: Permitirá al administrador registrar un nuevo usuario, llenando los campos Nombre, Apellidos, Nombre de usuario, Conraseña y Rol.
Observaciones: El administrador deberá estar autenticado en el sistema. Los campos Nombre, Apellidos, Nombre de usuario y Contraseña son obligatorios.
Interfaz: 

2.3.2. Estimación de esfuerzo por Historia de Usuario

En el ámbito del desarrollo ágil, es fundamental establecer un marco claro para la estimación del esfuerzo requerido en la implementación de las historias de usuario. En este proceso, los programadores utilizan una medida conocida como "punto", donde un punto representa una semana ideal de trabajo. Generalmente, las historias de usuario se valoran entre 1 y 3 puntos. Para llevar a cabo esta estimación, se traduce el esfuerzo en días, asegurando que la duración estimada oscile entre 1 y 21 días para que el valor total de la historia no supere los 3 puntos (Wells, 1999).

Tabla 2.5. Estimación de esfuerzo por historia de usuario.

Historia de Usuario	Puntos estimados
Autenticar usuario	3
Registrar usuario	3
Registrar recurso	3

Historia de Usuario	Puntos estimados
Modificar recurso	2
Registrar servicio	3
Modificar servicio	2
Ofrecer servicio	3
Visualizar ganancia total en una fecha determinada	3
Visualizar gasto total de recursos en una fecha determinada	3
Visualizar servicios ofrecidos en una fecha determinada	3
Eliminar recurso	2
Listar recursos	2
Buscar recurso	2
Eliminar servicio	2
Listar servicios	2
Buscar servicio	2
Cerrar sesión	2
Modificar usuario	2
Listar usuarios	2
Eliminar usuario	2
Seleccionar todos los usuarios	1
Deseleccionar todos los usuarios	1
Eliminar todos los usuarios seleccionados	1
Seleccionar todos los recursos	1
Deseleccionar todos los recursos	1
Eliminar todos los recursos seleccionados	1
Seleccionar todos los servicios	1
Deseleccionar todos los servicios	1
Eliminar todos los servicios seleccionados	1

2.3.3. Desarrollo del plan de iteraciones

El desarrollo del plan de iteraciones se caracteriza por su enfoque en la entrega incremental de software funcional y de alta calidad, mediante ciclos de desarrollo cortos y regulares. La planificación de iteraciones en XP está orientada a maximizar la adaptabilidad del proyecto, permitiendo que el equipo se ajuste de manera rápida y continua a las necesidades cambiantes del cliente. Cada iteración tiene una duración típica de no más de tres semanas. Durante estas, el equipo de desarrollo se enfoca en implementar un conjunto específico de historias de usuario priorizadas, las cuales representan funcionalidades descritas en términos de valor para el usuario final. Este conjunto se selecciona en colaboración con el cliente, quien es responsable de determinar la prioridad de las historias en función del valor de negocio y las necesidades del proyecto (FasterCapital, 2024).

En la presente solución, la planificación se enfoca en la implementación progresiva de funcionalidades clave para la gestión de los recursos y servicios en el taller. Para organizar el desarrollo, se asignaron niveles de prioridad a cada historia de usuario, lo cual permitió estructurar el proyecto en 3 iteraciones, con una duración de 57 días, que representan 8 semanas aproximadamente.

Tabla 2.6. Plan de duración de las iteraciones

Iteración	Historias de usuario		Duración (semanas)
1	1	Autenticar de usuario	28.0
	2	Registrar usuario	
	3	Registrar recurso	
	4	Modificar recurso	
	5	Registrar servicio	
	6	Modificar servicio	
	7	Ofrecer servicio	
	8	Visualizar ganancia total en una fecha determinada	
	9	Visualizar gasto total de recursos en una fecha determinada	
	10	Visualizar servicios ofrecidos en una fecha determinada	
2	11	Eliminar recurso	20.0
	12	Listar recursos	
	13	Buscar recursos	
	14	Eliminar servicio	
	15	Listar servicios	
	16	Buscar servicios	
	17	Cerrar sesión	
	18	Modificar usuario	
	19	Eliminar usuario	
	20	Listar usuarios	
3	21	Seleccionar todos los recursos	9.0
	22	Deseleccionar todos los recursos	
	23	Eliminar todos los recursos seleccionados	
	24	Seleccionar todos los servicios	
	25	Deseleccionar todos los servicios	
	26	Eliminar todos los servicios seleccionados	
	27	Seleccionar todos los usuarios	
	28	Deseleccionar todos los usuarios	
	29	Eliminar todos los usuarios seleccionados	
Total			**57.0**

- Iteración 1: Durante esta iteración, se implementan las historias de usuario con mayor prioridad. Al

finalizar, se obtendrán algunas de las funcionalidades básicas del sistema.

- Iteración 2: En esta fase, se continúan implementando las historias de usuario restantes con mayor prioridad, así como las de prioridad media.

- Iteración 3: Se lleva a cabo la implementación de las historias de usuario de prioridad baja.

2.3.4. Plan de entrega

El siguiente plan de entrega detalla la cantidad de historias de usuario que se completarán en cada iteración, así como las fechas de inicio y finalización correspondientes. Este proporciona una propuesta del progreso estimado a lo largo del proyecto.

Tabla 2.7. Plan de iteraciones.

Entregable	**Iteración 1**	**Iteración 2**	**Iteración 3**
Cantidad de HU	10	9	10
Fecha de inicio	17/09/24	15/10/24	06/11/24
Fecha de entrega	14/10/24	05/11/24	11/11/24

2.4. Fase II: Diseño

El diseño sigue rigurosamente el principio MS (mantenlo sencillo). Este enfoque se traduce en que el diseño guía la implementación de las historias conforme se desarrollan, evitando la creación de funcionalidades adicionales que no sean necesarias en ese momento. XP promueve el uso de tarjetas CRC (Clase-Responsabilidad-Colaborador) como una herramienta eficaz para conceptualizar el software dentro de un contexto orientado a objetos, facilitando la identificación y organización de las clases relevantes para el incremento actual del sistema. Además, fomenta el rediseño continuo, permitiendo que el diseño evolucione a medida que avanza la construcción del sistema, lo que contribuye su adaptación a los cambios necesarios. En este sentido, el diseño se considera un artefacto en transición que debe ser modificado constantemente para mejorar su efectividad y responder a los desafíos que surgen durante el desarrollo (Pressman, 2010).

2.4.1. Tarjetas Clase - Responsabilidad - Colaborador (CRC)

Las tarjetas CRC (Clase-Responsabilidad-Colaborador) son una herramienta utilizada en el diseño de sistemas orientados a objetos para definir y organizar clases. Cada tarjeta representa una clase y se divide en tres secciones: **Clase**, que describe el nombre de la clase; **Responsabilidad**, que especifica sus funciones principales; y **Colaborador**, que identifica otras clases con las que interactúa para cumplir sus responsabilidades. Estas facilitan la comprensión del diseño del sistema, permitiendo visualizar claramente las responsabi-

lidades y relaciones de las clases. Son útiles en etapas tempranas del desarrollo de software para iterar rápidamente en el diseño (Creately, 2024). A continuación, se muestra una de las tarjetas CRC definidas para la implementación de la solución, las restantes pueden ser consultadas en los anexos de la investigación (Ver A.4).

Tabla 2.8. Tarjeta CRC # 1

Tarjeta CRC	
Clase: Service	
Responsabilidad	**Colaboración**
• create(): Se utiliza para instanciar y persistir un nuevo objeto en la base de datos. • save(): Guarda el estado actual del objeto en la base de datos, ya sea creando un nuevo registro o actualizando uno existente. • get(): Permite recuperar un único objeto de la base de datos que coincide con los criterios especificados, lanzando una excepción si no se encuentra ninguno. • all(): Devuelve un conjunto de todos los objetos de la clase almacenados en la base de datos. • filter(): Permite consultar la base de datos y devuelve un conjunto de objetos que cumplen con los criterios de filtrado especificados. • delete(): Elimina el objeto de la base de datos, asegurando que ya no esté disponible en futuras consultas.	• ServiceResource • WorkerService

2.5. Arquitectura del sistema

La arquitectura de software de un sistema es el conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos y propiedades de ambos (Bass; Clements y Kazman, 2012). Esta puede ser entendida como la base sobre la cual se construyen las aplicaciones, proporcionando un marco que guía las decisiones de diseño y desarrollo a lo largo del ciclo de vida del software. Incluye elementos clave como los componentes, que son las unidades funcionales que realizan tareas específicas dentro del sistema; las conexiones, que representan las interacciones entre estos componentes;

y los estilos arquitectónicos, que son patrones generales que guían la organización del sistema, como la arquitectura en capas o basada en microservicios (Richards y Ford, 2020).

La importancia de una buena arquitectura radica en su capacidad para facilitar el mantenimiento, la escalabilidad y la adaptabilidad del sistema, permitiendo a los desarrolladores realizar cambios y ampliaciones sin comprometer la integridad del mismo (Ken, 2023). El diseño arquitectónico se interesa por entender cómo debe organizarse un sistema y cómo tiene que diseñarse su estructura global. Es la primera etapa en el proceso de diseño del software y constituye el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos.

Considerando la descripción de la propuesta de solución y las tecnologías elegidas, se sugiere implementar una arquitectura que facilite la construcción de sistemas escalables y flexibles, garantizando una clara separación de preocupaciones entre la interfaz de usuario y la lógica de negocio, por lo que se decide utilizar la arquitectura Cliente-Servidor, que permite distribuir efectivamente las responsabilidades entre los componentes del sistema.

La arquitectura Cliente-Servidor es un modelo de diseño de software en el que las aplicaciones se dividen en dos componentes principales: clientes y servidores. Esta separación facilita la gestión de recursos y permite que ambos interactúen de manera eficiente (Lizama; Kindley; Morales y Gonzales, 2016).

El cliente es la parte de las aplicaciones que inicia las peticiones de servicios hacia el servidor usando la red como mecanismo de comunicación. Generalmente, se ejecuta en la máquina del usuario y puede tomar varias formas, como un navegador web, aplicaciones de escritorio o móviles. Su principal función es enviar peticiones al servidor y recibir las respuestas adecuadas. Los clientes suelen presentar una interfaz de usuario que permite a las personas interactuar con los recursos y servicios proporcionados por el servidor (ibíd.).

El servidor es el componente que responde a las solicitudes que recibe del cliente. Este almacena y gestiona los recursos, que pueden incluir bases de datos, archivos y aplicaciones. La capacidad de procesamiento del servidor es crucial, ya que debe manejar múltiples peticiones simultáneamente y devolver respuestas activas. Entre los roles de los servidores se encuentran tareas como el almacenamiento de la información, el procesamiento de datos y el control de las reglas de negocio (ibíd.).

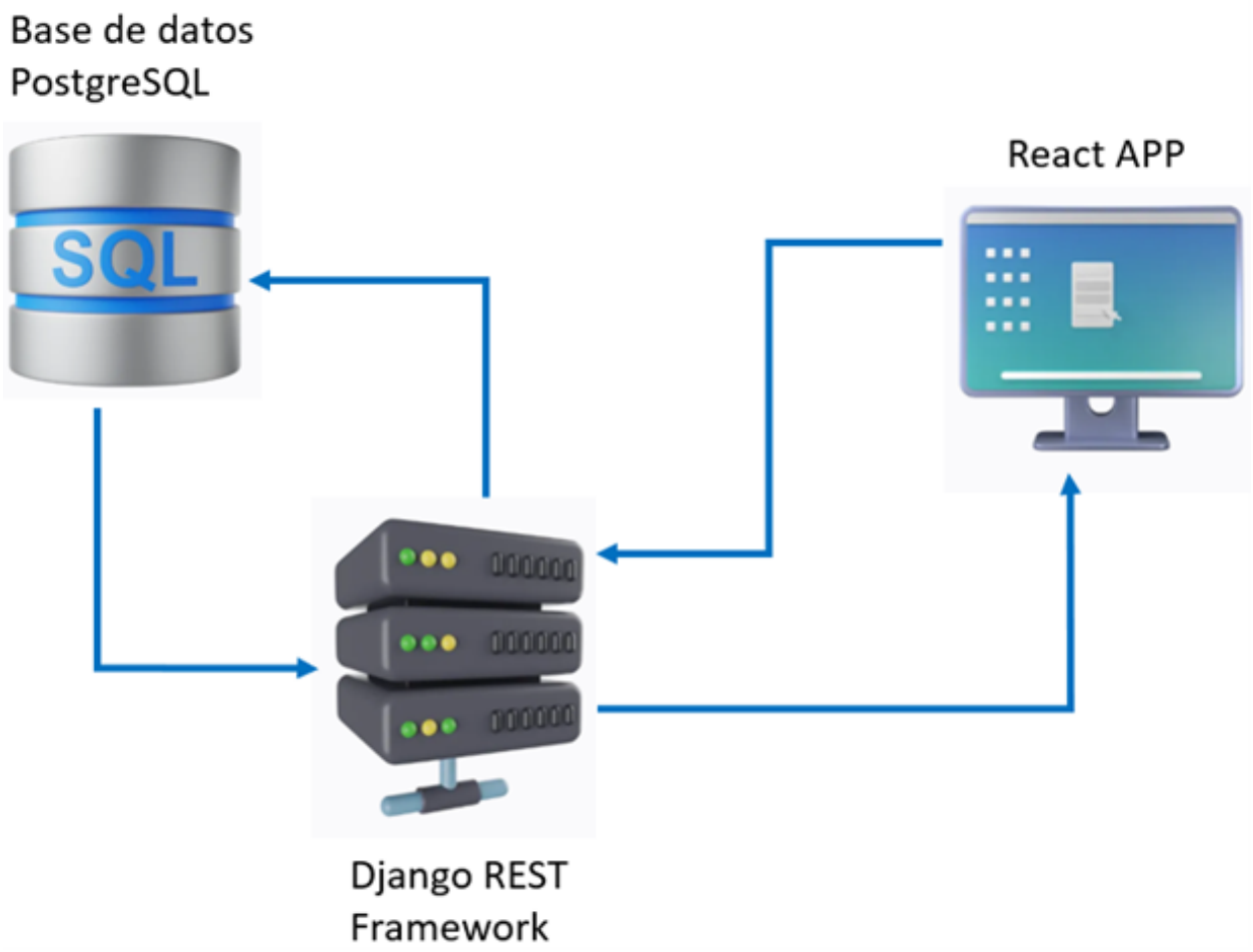


Figura 2.1. Representación de la Arquitectura Cliente-Servidor. [Fuente: elaboración propia].

La arquitectura Cliente-Servidor presentada se compone de los siguientes módulos clave: Django con Django REST Framework como backend , PostgreSQL como base de datos, y React como frontend. Cada uno de estos componentes desempeña un papel específico en la gestión de datos, en la lógica de negocio y en la interacción del usuario final con el sistema.

- Django REST Framework: Este módulo constituye el núcleo del *backend* y se encarga de manejar las solicitudes de la API[2]. Django REST Framework es una extensión de Django especializada en facilitar la construcción de aplicaciones web. Proporciona una interfaz completa que incluye vistas reutilizables y recursos predefinidos para simplificar la manipulación de datos y el control de acceso mediante operaciones HTTP, como *POST* y *GET*. Además, integra una interfaz de administración que permite realizar pruebas directas de estas operaciones, facilitando la depuración y validación del

[2]API (*Application Programming Interface* o interfaz de programación de aplicaciones) es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas (Fernandez, 2019).

sistema. Al ser un *framework* de alto nivel en Python, proporciona una estructura modular y escalable para desarrollar APIs Web robustas y seguras.

- PostgreSQL: Esta base de datos relacional es el sistema de almacenamiento de datos en la aplicación. PostgreSQL, como base de datos avanzada y compatible con múltiples tipos de datos y operaciones complejas, almacena toda la información relevante de la aplicación, desde datos de usuarios hasta configuraciones específicas. Django interactúa con PostgreSQL a través de un sistema ORM *(Object-Relational Mapping)*, que permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sin necesidad de escribir consultas SQL explícitas. Esta capa de abstracción optimiza el flujo de trabajo, al permitir que el desarrollador gestione y manipule los datos directamente en Python, manteniendo una integración fluida y coherente con el resto del *backend*.

- React: Como interfaz *frontend*, se encarga de la experiencia de usuario y de la comunicación con el *backend* mediante la API RESTful proporcionada por Django. A través de axios[3], puede enviar y recibir datos de forma no simultánea mediante métodos como *GET*, *POST*, *PUT* y *DELETE*. Esta funcionalidad permite a la aplicación React solicitar información, enviar datos de usuario o actualizar el estado de la aplicación en tiempo real sin recargar la página, mejorando la interactividad y fluidez de la experiencia del usuario. Al separar la lógica de presentación del manejo de datos, ofrece una estructura de componentes que simplifica la gestión del estado de la interfaz y la sincronización de datos con el *backend*.

2.6. Patrón arquitectónico

Un patrón arquitectónico en software es una guía estructural que aborda problemas recurrentes en el diseño de sistemas complejos. Define la organización de los componentes y la forma en que estos interactúan para cumplir con los objetivos de rendimiento, escalabilidad, flexibilidad y mantenibilidad. Ofrece una solución de diseño general que puede aplicarse en diversas situaciones, proporcionando un marco reutilizable para enfrentar desafíos comunes en la arquitectura de software (Murillo, 2023).

Uno de los patrones arquitectónicos más conocidos en la industria es el Modelo-Vista-Controlador. Este patrón separa presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí. El componente Modelo maneja los datos del sistema y las operaciones asociadas a esos datos. El componente Vista define y gestiona cómo se presentan los datos al usuario. El componente Controlador dirige la interacción del usuario (por ejemplo, teclas oprimidas, clics del mouse, etcétera) y pasa estas interacciones a Vista y Modelo (Sommerville, 2011).

[3]Axios es una biblioteca popular de JavaScript que se utiliza para hacer solicitudes HTTP desde el *frontend* a un servidor. Es comúnmente utilizada en aplicaciones React para interactuar con APIs RESTful (Halliday y Gorton, 2021).

La actual propuesta se basa en este patrón arquitectónico. Esta elección se justifica porque al permitir una clara delimitación de funciones, el patrón MVC promueve la reutilización del código, ya que los componentes son independientes y pueden ser modificados sin afectar al resto del sistema. Además, la separación de la lógica de negocio, la presentación y la interacción del usuario simplifica las actualizaciones y facilita la incorporación de nuevas características.

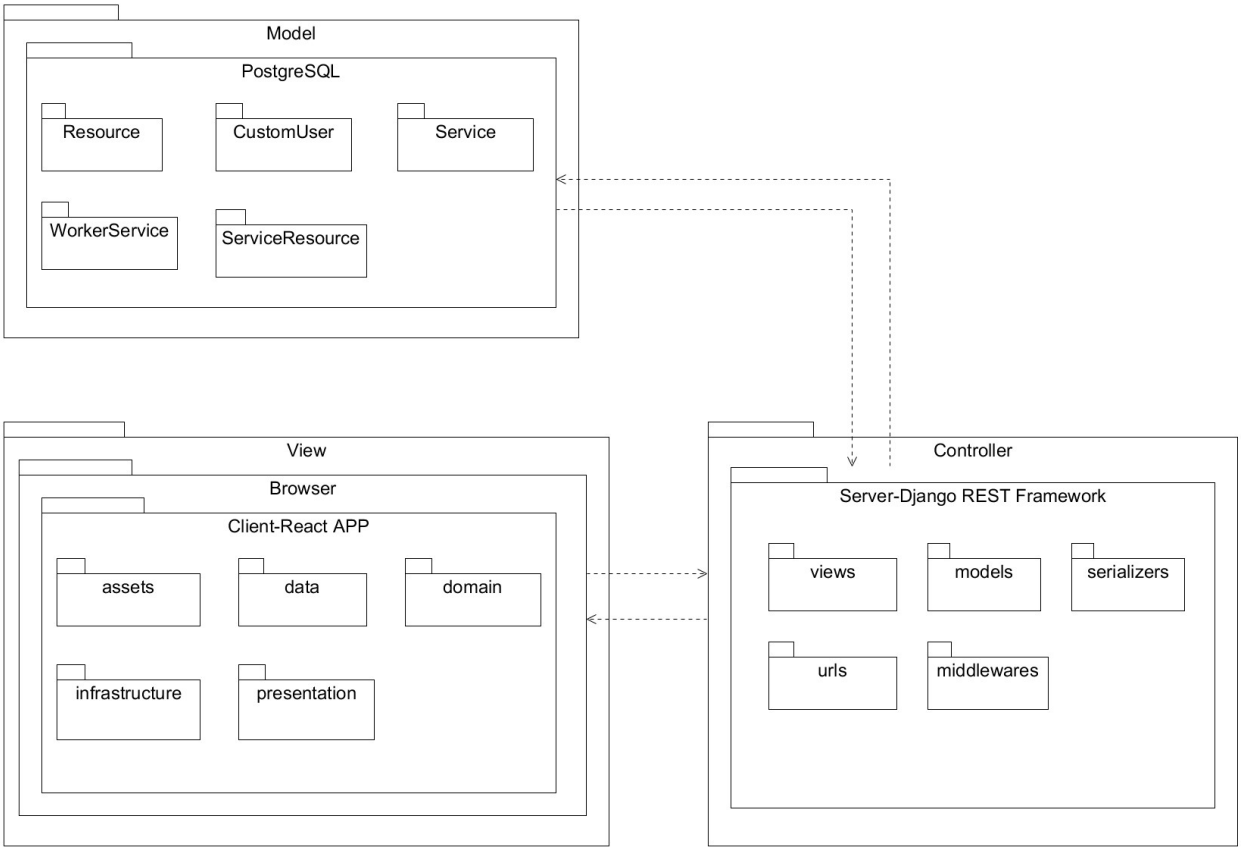


Figura 2.2. Representación del patrón arquitectónico Modelo-Vista-Controlador [Fuente: elaboración propia].

- PostgreSQL se encarga del Modelo, gestionando los datos de la aplicación como usuarios personalizados, recursos y servicios. Organiza estos datos en tablas relacionales, facilitando el almacenamiento y acceso a los datos de forma estructurada. A través del ORM de Django en el controlador, se realizan operaciones como consultas, inserciones y actualizaciones de datos para que el sistema mantenga la consistencia y la integridad de la información.

- El Cliente en React actúa como la Vista, permitiendo a los usuarios interactuar con la aplicación a través del navegador. Este se comunica con el *backend* (Django) usando HTTP/HTTPS a través de los puertos 80/443. La estructura incluye componentes, servicios y páginas organizados en módulos

como *assets*, *data*, *domain*, *infrastructure* y *presentation*, que estructuran la lógica de presentación y la interfaz de usuario. Los componentes de React envían solicitudes al *backend* para obtener datos o realizar acciones en la aplicación, para luego renderizar la información recibida en la interfaz de usuario, proporcionando una experiencia interactiva y dinámica a los usuarios.

- El servidor Django REST Framework en Python actúa como el Controlador, accediendo a la base de datos para gestionar los datos de la aplicación y procesar las solicitudes provenientes del cliente. Este servidor utiliza el protocolo HTTP/HTTPS a través del puerto 3000 para recibir peticiones, procesarlas, interactuar con el modelo en PostgreSQL y devolver las respuestas correspondientes al cliente en React. Dentro de este controlador, las *URLs* define las rutas de la API que determinan qué vista ejecutar según el tipo de solicitud; las *Views* (vistas) procesan la lógica de negocio y manejan cada solicitud que llega al sistema; los *Serializers* convierten los datos del modelo a formato JSON para facilitar la comunicación entre el *frontend* y el *backend*; y los *Middlewares* gestionan tareas como la autenticación y autorización, asegurando que el flujo de solicitudes sea seguro.

2.7. Patrones de diseño

Los patrones de diseño, o *design patterns*, constituyen una solución general, reutilizable y aplicable a diversos problemas de diseño de software. Son plantillas que identifican problemas comunes en los sistemas y proporcionan soluciones adecuadas a dificultades generales que los desarrolladores han enfrentado durante mucho tiempo, refinadas a través de prueba y error. Estos patrones ofrecen un enfoque estructurado y reutilizable para resolver situaciones recurrentes en el desarrollo de software. Facilitan la comunicación entre desarrolladores y el intercambio de soluciones perdurables que han demostrado su éxito previamente (Canelo, 2020).

Los patrones de diseño identificados en la propuesta de solución incluyen aquellos del conjunto GRASP (*General Responsibility Assignment Software Patterns* o Patrones Generales de Asignación de Responsabilidades en Software) y del conjunto GoF (*Gang of Four* o Grupo de los cuatro). Esta selección se basa en la importancia de ambos grupos para la asignación de responsabilidades y la resolución de problemas repetitivos en el desarrollo de software. Los patrones GRASP aportan directrices para una distribución adecuada de tareas entre los distintos componentes del sistema, mientras que los patrones GoF ofrecen soluciones estandarizadas para problemas comunes de diseño.

2.7.1. Patrones GRASP

Los patrones GRASP *(General Responsibility Assignment Software Patterns)* son un conjunto de directrices de diseño orientadas a la asignación de responsabilidades en el desarrollo de software. Propuestos por Craig Larman, estos patrones se centran en la distribución de funciones y responsabilidades entre los distintos

componentes del sistema, abordando aspectos como la cohesión, el acoplamiento y la comunicación efectiva entre clases (Mora, 2003).

Patrón Creador

El patrón Creador sugiere que una clase que necesita instancias de otra clase debería ser responsable de crearlas, especialmente si tiene una relación cercana o una dependencia funcional. Este principio ayuda a organizar la creación de objetos, promoviendo un flujo claro y controlado en el proceso de instanciación y reduciendo dependencias innecesarias entre las clases del sistema (Tabares, 2010) .

Este patrón se evidencia a través de la relación entre la clase *ResourceRepositoryImpl* y los objetos de tipo *Resource* que maneja en sus métodos. En este caso, la clase *ResourceRepositoryImpl* sigue el patrón Creador porque es responsable de interactuar con instancias de *Resource* para su creación, actualización o eliminación en el sistema. Esta clase centraliza la gestión de recursos, manejando tanto la creación como la modificación y eliminación de estos objetos, lo que le otorga la responsabilidad de controlar el ciclo de vida de los *Resource* en el sistema.

src > data > repositories > ResourceRepositoryImpl.ts > ...

```ts
import { Resource } from "../../domain/entities/resource";
import api from "../../infrastructure/api/axiosConfig";
import { ResourceRepository } from "../../domain/repositories/ResourceRepository";

export const ResourceRepositoryImpl: ResourceRepository = {
  getResources: async (
    start: number,
    end: number,
    name: string
  ): Promise<{ data: { resources: Resource[] } }> =>
    await api.post("resources/", { start, end, name }),

  createResource: async (resource: Resource) =>
    await api.post("create_resource/", resource),

  updateResource: async (resource: Resource) =>
    await api.post("update_resource/", resource),

  deleteResource: async (id: number) =>
    await api.post("delete_resource/", { id: id }),

  deleteManyResources: async (ids: number[]) =>
    await api.post("delete_many_resources/", { ids: ids }),
};
```

Figura 2.3. Representación del patrón Creador [Fuente: elaboración propia].

Patrón Experto

El patrón Experto consiste en asignar responsabilidades a la clase que posee la mayor cantidad de información requerida para llevar a cabo una tarea específica. Este patrón se aplica para garantizar que cada clase se encargue de las operaciones que puede gestionar directamente, facilitando así la comprensión, mantenimiento y adaptación del sistema a posibles cambios futuros (Tabares, 2010).

Su uso se evidencia a través de la asignación de las responsabilidades relacionadas con la obtención, inserción, actualización y eliminación de recursos a la clase *ResourceRepositoryImpl*. Esta clase actúa como el experto en la manipulación de datos relacionados con los recursos (*Resource*), ya que posee o recibe la información necesaria para interactuar con los *endpoints*[4] de la API y manejar las operaciones CRUD sobre los recursos. Esto permite centralizar la lógica de negocio en un único punto, facilitando el mantenimiento y la extensión del sistema.

src > data > repositories > ResourceRepositoryImpl.ts > ...

```ts
import { Resource } from "../../domain/entities/resource";
import api from "../../infrastructure/api/axiosConfig";
import { ResourceRepository } from "../../domain/repositories/ResourceRepository";

export const ResourceRepositoryImpl: ResourceRepository = {
  getResources: async (
    start: number,
    end: number,
    name: string
  ): Promise<{ data: { resources: Resource[] } }> =>
    await api.post("resources/", { start, end, name }),

  createResource: async (resource: Resource) =>
    await api.post("create_resource/", resource),

  updateResource: async (resource: Resource) =>
    await api.post("update_resource/", resource),

  deleteResource: async (id: number) =>
    await api.post("delete_resource/", { id: id }),

  deleteManyResources: async (ids: number[]) =>
    await api.post("delete_many_resources/", { ids: ids }),
};
```

Figura 2.4. Representación del patrón Experto [Fuente: elaboración propia].

Patrón Bajo Acoplamiento

[4]Un *endpoint* o punto final es donde una API recibe solicitudes. Para la mayoría de los servicios, estos puntos finales son URL, como las que se utilizan para navegar a un sitio web (Yusuf y Bryan, 2023).

El patrón Bajo Acoplamiento es una directriz de diseño que busca reducir la dependencia entre clases, permitiendo que los cambios en una clase afecten mínimamente a otras. Este patrón fomenta la flexibilidad del sistema, ya que cada clase interactúa con las demás solo cuando es estrictamente necesario (Tabares, 2010).

El principio de Bajo Acoplamiento se manifiesta a través de la separación de responsabilidades entre el serializador y el modelo de usuario. La clase *UserSerializer* se encarga de la serialización y deserialización de los datos relacionados con el usuario, sin involucrarse en los detalles internos del modelo *CustomUser*. Este enfoque garantiza que el serializador funcione independiente del modelo, lo que permite que ambos componentes puedan modificarse sin afectar al otro.

Además, el manejo de la contraseña está desacoplado del proceso de creación del usuario. En lugar de que el serializador gestione directamente los detalles del cifrado de la contraseña, utiliza el método *set_password()* del modelo, delegando así esta tarea a la lógica interna del modelo *CustomUser*. Esto reduce la dependencia entre el serializador y los detalles de implementación del modelo, manteniendo un acoplamiento mínimo y promoviendo un diseño más modular y flexible.

```
10  class UserSerializer(serializers.ModelSerializer):
11      class Meta:
12          model = CustomUser
13          fields = ['id', 'username', 'password', 'first_name', 'last_name', 'role']
14
15      def create(self, validated_data):
16          password = validated_data.pop('password')
17          user = CustomUser(**validated_data)
18          user.set_password(password)
19          user.save()
20          return user
21
```

Figura 2.5. Representación del patrón Bajo Acoplamiento [Fuente: elaboración propia].

Patrón Alta Cohesión

El patrón Alta Cohesión es una media que determina cuán relacionadas y adecuadas están las responsabilidades de una clase, de modo que no se realice un trabajo colosal; una clase con baja cohesión realiza un trabajo excesivo, haciéndola difícil de comprender, reutilizar y conservar (ibíd.).

El patrón Alta Cohesión se refleja de la manera en que las vistas relacionadas con la gestión de usuarios están estructuradas. Cada función (*register, login, update_user, delete_user,* etc.) tiene una responsabilidad específica y bien delimitada, centrada exclusivamente en la manipulación de usuarios, como el registro, la

actualización, y la eliminación. Esto asegura que las tareas no se mezclen y cada función se enfoque en un propósito claro, lo que facilita el mantenimiento y la extensibilidad del código.

Al manejar las solicitudes HTTP de manera independiente en cada vista con decoradores como *@api_view* y *@permission_classes*, el diseño del código asegura que las funciones se mantengan cohesionadas, relacionadas directamente con la gestión de usuarios. Este enfoque promueve un código más claro, modular y fácil de mantener, con una separación clara de responsabilidades.

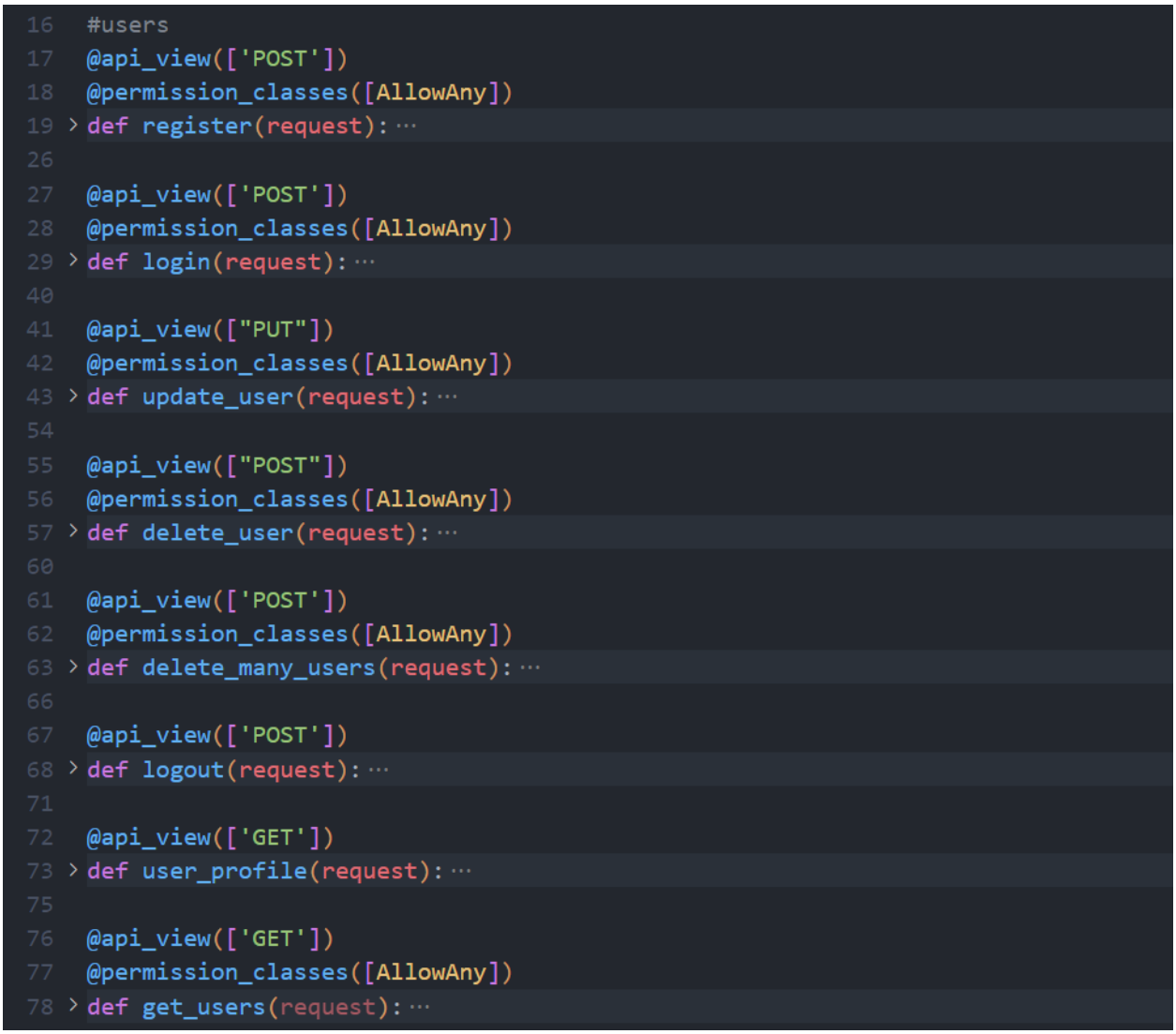


Figura 2.6. Representación del patrón Alta Cohesión [Fuente: elaboración propia].

2.7.2. Patrones GoF

Los patrones GoF (*Gang of Four*) son una serie de patrones de diseño establecidos por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides en el libro *"Design Patterns: Elements of Reusable Object Oriented Sofware"*. Estos patrones, ampliamente utilizados en el desarrollo de software, proporcionan soluciones estandarizadas a problemas recurrentes de diseño y están divididos en tres categorías: patrones de creación, estructurales y de comportamiento. La relevancia de los patrones GoF radica en su aplicabilidad a diversos contextos de diseño, ofreciendo soluciones probadas que mejoran la reutilización de código y el funcionamiento del software (Guerrero; Suarez y Gutierrez, 2013).

Patrón Compuesto

El concepto básico del patrón Compuesto (*Composite*) consiste en representar objetos simples y sus *containers* (o contenedores, también llamados colecciones en algunos lenguajes, o sea: grupos de objetos) en una clase abstracta para que puedan ser tratados uniformemente. Este tipo de estructura se conoce como jerarquía parte-todo, en la que un objeto es siempre, o una parte de un todo, o un todo compuesto por varias partes (IONOS, 2020).

El patrón Compuesto se manifiesta en la estructura de los componentes de React. Este patrón permite crear interfaces de usuario complejas mediante la composición de componentes más simples, manteniendo una manipulación uniforme. El componente *Modal* actúa como contenedor principal, y dentro de él se anidan otros elementos, como *div*, un botón para cerrar el modal y otros nodos React que se reciben a través de la propiedad *children*. Esta estructura jerárquica permite que el componente padre (*Modal*) gestione con uniformidad a sus componentes hijos, independientemente de su tipo o contenido.

Este patrón permite que Modal reciba y renderice otros componentes o elementos a través de *children*, lo que encapsula la idea central del patrón: manejar la composición de componentes sin alterar la lógica interna del componente contenedor. Esto facilita la creación de interfaces modulares y promueve una estructura jerárquica que simplifica tanto la composición como la manipulación de los elementos en React.

src > presentation > components > modal > Modal.tsx > ...

```tsx
import { createPortal } from "react-dom";

import icons from "../../../utils/icons";

import "./modal.css";

interface ModalProps {
  children: React.ReactNode;
  modal: any;
  onClose?: () => void;
}

export const Modal: React.FC<ModalProps> = ({ children, modal, onClose }) => {
  return createPortal(
    <div className={`modal ${modal.isClosing ? "modal-closing" : ""} `}>
      <div
        className={`modal-container ${modal.isClosing ? "modal-closing" : ""}`}
      >
        <button className="close-btn" onClick={onClose ? onClose : modal.close}>
          <i className={icons.xmark}></i>
        </button>
        {children}
      </div>
    </div>,
    document.getElementById("modal-overlay") as HTMLElement
  );
};
```

Figura 2.7. Representación del patrón Compuesto [Fuente: elaboración propia].

Patrón Método de Fabricación

El patrón Método de Fabricación (*Factory Method*) permite que una clase delegue la creación de objetos a sus subclases, permitiendo diferentes implementaciones sin modificar la lógica de la clase principal. Este patrón define dos roles: creadores (fábricas) y productos (objetos creados). Al hacer el método de creación abstracto, cada subclase puede personalizar la creación de sus productos, lo que es útil en aplicaciones que requieren múltiples variantes de un mismo producto (Gorkovenko, 2023).

El uso de este patrón se evidencia en cómo el *hook*[5] personalizado *useValidator* funciona como una fábrica para crear validadores específicos según el campo del formulario que se está validando (nombre, cantidad, o precio). Cada vez que se invoca *useValidator*, este recibe las reglas de validación necesarias y produce un validador adecuado para el tipo de dato correspondiente. Por ejemplo, el *nameValidator* verifica si el campo

[5]Los *Hooks* son funciones que te permiten enganchar el estado de React y el ciclo de vida desde componentes de función.

de nombre está vacío, el *quantityValidator* asegura que la cantidad sea un número entero, y el *priceValidator* valida que el precio sea un número válido.

Este enfoque, en el cual la creación de objetos, en este caso, los validadores, se delega a una función centralizada que configura sus parámetros. Gracias a esta abstracción, el código es fácil de extender, permitiendo añadir validadores nuevos simplemente pasando nuevas reglas a *useValidator* sin necesidad de modificar su implementación interna. Esto permite que la lógica de creación de validadores esté desacoplada del código que los utiliza, mejorando así la flexibilidad y la reutilización del código.

```
40  const nameValidator = useValidator(() => {
41    const name = formState.name.trim();
42    if (name === "") {
43      return "Debe insertar un nombre";
44    }
45  });
46
47  const quantityValidator = useValidator(() => {
48    if ("" + formState.quantity.trim() === "")
49      return "Debe insertar una cantidad";
50    const quantity = Number(formState.quantity);
51    if (isNaN(quantity)) return "La cantidad insertada no es ún numero";
52    if (!Number.isInteger(quantity))
53      return "La cantidad debe ser un número entero";
54  });
55
56  const priceValidator = useValidator(() => {
57    if (formState.price.trim() === "") return "Debe insertar un precio";
58    let price = Number(formState.price);
59    if (isNaN(price)) {
60      return "El precio insertado no es un numero";
61    }
62  });
```

Figura 2.8. Representación del patrón Método de Fabricación [Fuente: elaboración propia].

Patrón Fachada

El patrón Fachada (*Facade*) tiene la característica de ocultar la complejidad de interactuar con un conjunto de subsistemas proporcionando una interface de alto nivel, la cual se encarga de realizar la comunicación con todos los subsistemas necesarios. Así, los usuarios o aplicaciones pueden acceder a funcionalidades completas sin preocuparse por los detalles internos de cada subsistema. Este patrón es útil cuando varios sistemas deben coordinarse para completar tareas específicas y ayuda a reducir la complejidad al ocultar

procesos internos (Blancarte, 2016).

El patrón Fachada se observa en el objeto *ResourceRepositoryImpl*, que proporciona una interfaz unificada y simplificada para interactuar con varios *endpoints* relacionados con el manejo de recursos (*Resources*). Este objeto actúa como una fachada que abstrae la complejidad de las interacciones HTTP mediante métodos como *getResources*, *createResource*, *updateResource*, *deleteResource*, y *deleteManyResources*. Cada método se encarga de llamar a los endpoints correspondientes, proporcionando una API coherente y fácil de usar para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre recursos.

La función de este patrón es delegar la complejidad de las solicitudes HTTP al objeto api, encapsulando en *ResourceRepositoryImpl* la lógica necesaria para gestionar las interacciones de red. Esto permite que el cliente acceda de forma simple a las operaciones sin tener que manejar directamente la lógica de comunicación subyacente. Además, al centralizar las interacciones HTTP en esta clase, se facilita el mantenimiento y la evolución del sistema; cualquier cambio en los *endpoints* o en la lógica puede gestionarse dentro de la fachada sin afectar al resto del código.

src > data > repositories > ResourceRepositoryImpl.ts > ...

```typescript
import { Resource } from "../../domain/entities/resource";
import api from "../../infrastructure/api/axiosConfig";
import { ResourceRepository } from "../../domain/repositories/ResourceRepository";

export const ResourceRepositoryImpl: ResourceRepository = {
  getResources: async (
    start: number,
    end: number,
    name: string
  ): Promise<{ data: { resources: Resource[] } }> =>
    await api.post("resources/", { start, end, name }),

  createResource: async (resource: Resource) =>
    await api.post("create_resource/", resource),

  updateResource: async (resource: Resource) =>
    await api.post("update_resource/", resource),

  deleteResource: async (id: number) =>
    await api.post("delete_resource/", { id: id }),

  deleteManyResources: async (ids: number[]) =>
    await api.post("delete_many_resources/", { ids: ids }),
};
```

Figura 2.9. Representación del patrón Fachada [Fuente: elaboración propia].

Patrón Cadena de Responsabilidad

El patrón Cadena de Responsabilidad (*Chain of Responsibility*) es un patrón que se distingue por su versatilidad. Este enfoque es útil para distribuir tareas entre diferentes manejadores, aumentando la flexibilidad y reduciendo dependencias directas entre clases. Resuelve problemas fácilmente donde la herencia no puede. Apoyándose de una estructura en forma de cadena donde una secuencia de objetos trata de atender una petición (Galicia, 2019).

Este patrón permite que cada función de *middleware*[6] actúe como un manejador dentro de una cadena, con la responsabilidad de validar un aspecto específico de la solicitud. Los decoradores *@api_view([POST])* y *@permission_classes([AllowAny])* configuran la vista para que acepte solicitudes *POST* y permitan el acceso a cualquier usuario. Dentro de la función *register*, se utiliza el *UserSerializer* para validar y procesar los datos de la solicitud. Si *serializer.is_valid()* falla, el flujo se detiene y se devuelve una respuesta HTTP 409 Conflict, evitando que las siguientes etapas (como la creación del usuario y del token) se ejecuten.

Proporciona una gestión modular de las solicitudes, facilitando la adición, eliminación o modificación de validadores sin afectar la lógica principal del controlador. Promueve el desacoplamiento entre componentes, resultando en un código más limpio y mantenible. La flexibilidad y reutilización de los validadores en diferentes contextos optimizan el proceso de desarrollo, permitiendo responder rápidamente a cambios en los requisitos y mejorar la escalabilidad de la aplicación.

```
17  @api_view(['POST'])
18  @permission_classes([AllowAny])
19  def register(request):
20      serializer = UserSerializer(data={k: v for k, v in request.data.items() if k != 'id'})
21      if serializer.is_valid():
22          user = serializer.save()
23          token = Token.objects.create(user=user)
24          return Response({'token': token.key, 'user': serializer.data})
25      return Response(status=status.HTTP_409_CONFLICT)
```

Figura 2.10. Representación del patrón Cadena de responsabilidad [Fuente: elaboración propia].

2.8. Modelo de datos

Si los requerimientos del software incluyen la necesidad de crear, ampliar o hacer interfaz con una base de datos, o si deben construirse y manipularse estructuras de datos complejas, el equipo del software tal vez elija crear un modelo de datos como parte del modelado general de los requerimientos (Pressman, 2010). El modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Este describe las estructuras de datos correspondiente al contenido del sistema, permite describir los elementos que intervienen

[6]El *middleware* es un software que se sitúa entre las aplicaciones y el sistema operativo en el que se ejecutan. Funciona como una capa que permite la comunicación y la administración de datos en las aplicaciones (Tenea, 2024).

y la forma en que se relacionan estos elementos entre sí. A continuación, se muestra el modelo de datos correspondiente a la solución propuesta.

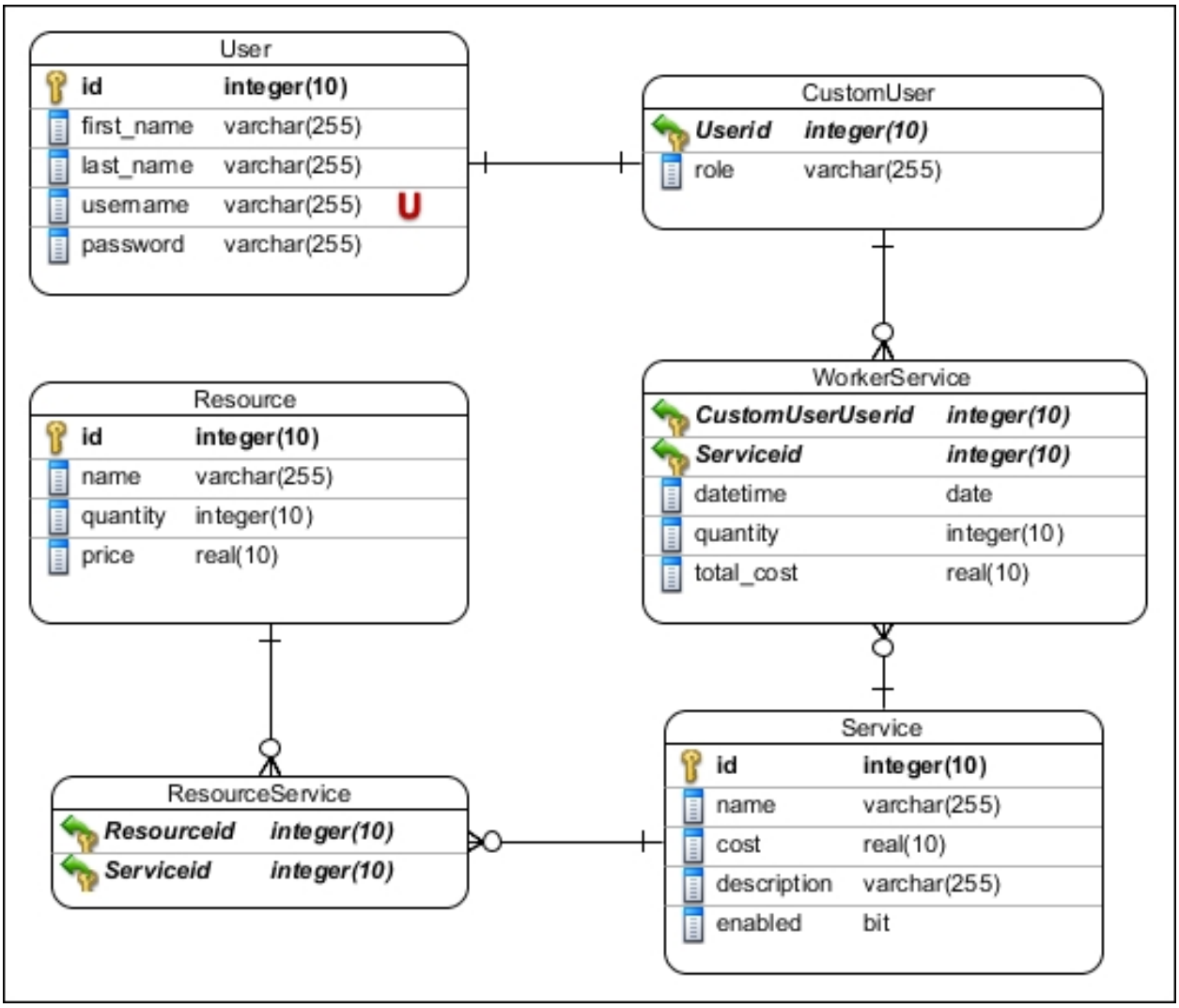


Figura 2.11. Diagrama Entidad-Relación [Fuente: elaboración propia].

El modelo de datos diseñado consta de seis tablas, cada una destinada a almacenar y gestionar la información de manera segura y estructurada. La tabla denominada *User* corresponde a un modelo predefinido del *framework* Django y desempeña un papel fundamental en el registro y la gestión de usuarios del sistema. Para extender las funcionalidades del modelo estándar de usuario, se incluye la tabla *CustomUser*, la cual incorpora atributos adicionales, como el rol del usuario, que puede clasificarse como trabajador o administrador, representando los distintos tipos de usuarios registrados en el sistema.

La tabla *Service* se encarga de almacenar información relativa a los servicios disponibles en el sistema,

mientras que los recursos materiales necesarios para la prestación de dichos servicios se gestionan en la tabla *Resources*. La relación entre servicios y recursos queda definida en la tabla intermedia *ResourceService*, donde se registran las asignaciones específicas de recursos a los servicios correspondientes. Por otro lado, la tabla *WorkerService* almacena los datos sobre los servicios realizados por los trabajadores, incluyendo información detallada como la fecha y hora de ejecución, la cantidad y el costo total asociado. Este diseño garantiza una estructura lógica que facilita la organización, seguridad y acceso a los datos, optimizando así la comprensión y gestión de la información dentro del sistema.

2.9. Conclusiones parciales

En este capítulo se analizaron diversos aspectos vinculados al diseño del sistema que se pretende desarrollar. Como resultado de este análisis, se arribó a las siguientes conclusiones:

- Se identificaron 29 requisitos funcionales y 12 no funcionales, y a partir de los primeros, se definieron 29 historias de usuario, lo que permitió organizar las funcionalidades esperadas del sistema en unidades concretas y medibles.

- Con el uso de las tarjetas CRC se definieron de manera precisa la colaboración entre los componentes del sistema.

- La elección de la arquitectura Cliente-Servidor combinada con el patrón Modelo-Vista-Controlador (MVC) permitió una separación clara entre la interfaz de usuario y la lógica de negocio.

- La incorporación de los patrones de diseño GRASP y GoF permitió organizar las responsabilidades de manera lógica y minimizar el acoplamiento entre componentes.

- Se generó el modelo de datos que permitió estructurar de manera lógica la información y garantizar una representación clara de las relaciones entre los elementos del sistema.

CAPÍTULO 3

IMPLEMENTACIÓN Y VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

El presente capítulo aborda los estándares de codificación empleados en la propuesta de solución, además de las tareas de ingeniería llevadas a cabo en cada una de las iteraciones. Se realizan pruebas unitarias y pruebas de aceptación para garantizar que las funcionalidades implementadas cumplan con los requisitos establecidos y que el software opere de manera efectiva en diferentes escenarios.

3.1. Fase III: Codificación

Después de que las historias han sido desarrolladas y de que se ha hecho el trabajo de diseño preliminar, el equipo no inicia la codificación, sino que desarrolla una serie de pruebas unitarias a cada una de las historias que se van a incluir en la entrega en curso (incremento de software). Una vez creada la prueba unitaria, el desarrollador está capacitado para centrarse en lo que debe implementarse para pasar la prueba. No se agrega nada extraño (MS). Una vez que el código está terminado, se le aplica de inmediato una prueba unitaria, con lo que se obtiene retroalimentación instantánea para los desarrolladores (Pressman, 2010).

3.1.1. Estándares de codificación

El establecimiento de estándares de codificación consiste en un conjunto de reglas y convenciones que determinan los formatos y estilos a utilizar en la redacción de código. La adopción de estas normas permite garantizar la legibilidad y uniformidad del código, lo que a su vez facilita su comprensión, mantenimiento y refactorización (Nekrasov, 2023).

En el contexto de XP, estos estándares juegan un papel crucial en la programación por parejas, uno de los pilares fundamentales de esta metodología. Mientras una persona en la pareja se enfoca en la implementación de los detalles técnicos del código, la otra verifica el cumplimiento de los estándares establecidos. A continuación, se presentan un conjunto de estándares basados en las convenciones de codificación propias de los lenguajes Python y TypeScript, adaptadas al marco de trabajo Django y a la biblioteca React respec-

tivamente.

Estándares de codificación para el lenguaje Python utilizando como marco de trabajo Django REST Framework:

1. Estructura de archivos y organización:
    - Cada archivo Python debe contener una sola clase principal o un conjunto de funciones relacionadas.
    - Los módulos deben agruparse de manera lógica, siguiendo la estructura recomendada de Django:
        - *models.py*: Para definiciones de modelos.
        - *views.py*: Para vistas relacionadas con la lógica de negocio.
        - *serializers.py*: Para serialización de datos.
        - *urls.py*: Para configuraciones de rutas.
2. Estructura de archivos y organización:
    - Identación de 4 espacios.
    - Uso de nombres en snake_case para funciones y variables.
    - Clases en *CamelCase*.
    - Documentar cada clase, método y función.
3. Convenciones de formato:
    - Un solo espacio antes y después de los operadores (==, and, or, etc.).
    - Añadir una coma final después del último elemento en listas y diccionarios multilínea.
4. Gestión de propiedades de clases:
    - Todas las propiedades de las clases del modelo deben ser privadas y el acceso debe realizarse a través de métodos *get* y *set*.
5. Seguridad y configuración:
    - Utilizar variables de entorno para datos sensibles (claves API, contraseñas).
    - Estandarizar configuraciones en archivos separados para desarrollo, pruebas y producción.
6. Estructura de archivos y organización:
    - Identación de 4 espacios.
    - Uso de nombres en *snake_case* para funciones y variables.
    - Clases en *CamelCase*.
    - Documentar cada clase, método y función.

Estándares de codificación para el lenguaje TypeScript utilizando la biblioteca React:

1. Estructura de componentes y archivos:

- Cada componente debe tener su propio directorio que incluya el archivo del componente (*ComponentName.tsx*), estilos asociados (*ComponentName.module.css* o CSS-in-JS), archivos de prueba (*ComponentName.test.tsx*) y usar una estructura jerárquica para organizar componentes por función o sección de la interfaz.

2. Convenciones de nombres:
   - Nombres de componentes en *PascalCase*.
   - Nombres de funciones y variables en *camelCase*.
   - Archivos con nombres descriptivos en *CamelCase*.
3. Estilo de código y formato:
   - Usar siempre corchetes incluso si el bloque contiene una sola línea.
   - Comentar bloques complejos usando comentarios multilínea (/** */).
4. Uso de TypeScript:
   - Definir explícitamente tipos para todas las variables, propiedades de componentes y funciones.
   - Usar *interface* para definir las *props* de los componentes y *type* para datos más simples.
5. Buenas prácticas:
   - Los estados deben manejarse con *useState* o *useReducer* en lugar de variables globales innecesarias.
   - Reutilizar lógica compartida mediante *hooks* personalizados.

Al aplicar estándares de manera consistente, incluso en proyectos individuales como es el caso de la actual propuesta, se fomenta la propiedad colectiva del código en el caso de futuras colaboraciones o ampliaciones del equipo. Además, un código uniforme no solo mejora la calidad técnica del software, sino que también aceleran la resolución de problemas y promueve prácticas sostenibles de desarrollo.

3.1.2. Tareas de ingeniería

En el contexto documental de la Programación Extrema (XP), esta metodología propone dividir las historias de usuario (HU) en tareas de ingeniería (TI) con el fin de simplificar y estructurar la implementación del sistema que se está desarrollando. Estas tareas de ingeniería actúan como unidades más pequeñas y manejables que descomponen los objetivos de alto nivel planteados en las historias de usuario, permitiendo un enfoque más detallado y organizado para su ejecución (Jummp, 2011).

La descomposición de las HU en TI facilita la planificación, distribución de responsabilidades y seguimiento del trabajo. Cada tarea se documenta de manera precisa mediante una ficha que incluye información clave, como su identificador único, el vínculo con la historia de usuario correspondiente, el tipo de tarea, fechas de inicio y finalización, el equipo responsable y una descripción detallada. La presente investigación está estructurada en 3 iteraciones, donde quedaron definidas las historias de usuario correspondientes. Estas HU

fueron divididas en 29 tareas de ingeniería, de las que se muestran dos por cada iteración. Las restantes TI se encuentran en los anexos de la investigación (Ver A.5).

Iteración 1: Se realizaron un total de 10 tareas de ingeniería:

HU 1: Autenticar usuario.

- TI 1: Autenticar usuario.

HU 2: Registrar usuario.

- TI 2: Registrar usuario.

HU 3: Registrar recurso.

- TI 3: Registrar recurso.

HU 4: Modificar recurso.

- TI 4: Modificar recurso.

HU 5: Registrar servicio.

- TI 5: Registrar servicio.

HU 6: Modificar servicio.

- TI 6: Modificar servicio.

HU 7: Ofrecer servicio.

- TI 7: Ofrecer servicio.

HU 8: Visualizar ganancia total en una fecha determinada.

- TI 8: Visualizar ganancia total en una fecha determinada.

HU 9: Visualizar gasto total de recursos en una fecha determinada.

- TI 9: Visualizar gasto total de recursos en una fecha determinada.

HU 10: Visualizar servicios ofrecidos en una fecha determinada.

- TI 10: Visualizar servicios ofrecidos en una fecha determinada.

Tabla 3.1. Tarea de desarrollo # 1

Tarea	
Número de tarea: 1	**Número de Historia de usuario:** 1
Nombre de la tarea: Autenticar usuario	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 17 de septiembre de 2024	**Fecha de fin:** 19 de septiembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite al usuario autenticarse en el sistema.	

Tabla 3.2. Tarea de desarrollo # 2

Tarea	
Número de tarea: 2	**Número de Historia de usuario:** 2
Nombre de la tarea: Registar usuario	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 20 de septiembre de 2024	**Fecha de fin:** 22 de septiembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite registrar un nuevo usuario en el sistema.	

Iteración 2: Se realizaron un total de 10 tareas de ingeniería:

HU 11: Eliminar recurso.

- TI 11: Eliminar recurso.

HU 12: Listar recursos.

- TI 12: Listar recursos.

HU 13: Buscar recursos.

- TI 13: Buscar recursos.

HU 14: Eliminar servicio.

- TI 14: Eliminar servicio.

HU 15: Listar servicios.

- TI 15: Listar servicios.

HU 16: Buscar servicios.

- TI 16: Buscar servicios.

HU 17: Cerrar sesión.

- TI 17: Cerrar sesión.

HU 18: Modificar usuario.

- TI 18: Modificar usuario.

HU 19: Eliminar usuario.

- TI 19: Eliminar usuario.

HU 20: Listar usuarios.

- TI 20:Listar usuarios.

Tabla 3.3. Tarea de desarrollo # 3

Tarea	
Número de tarea: 3	**Número de Historia de usuario:** 11
Nombre de la tarea: Eliminar recurso	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 15 de octubre de 2024	**Fecha de fin:** 17 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite al usuario autenticarse en el sistema.	

Tabla 3.4. Tarea de desarrollo # 4

Tarea	
Número de tarea: 4	**Número de Historia de usuario:** 12
Nombre de la tarea: Listar recursos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 18 de octubre de 2024	**Fecha de fin:** 19 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite visualizar los recursos en forma de lista.	

Iteración 3: Se realizaron un total de 9 tareas de ingeniería:

HU 21: Seleccionar todos los recursos.

- TI 21: Seleccionar todos los recursos.

HU 22: Deseleccionar todos los recursos.

- TI 22: Deseleccionar todos los recursos.

HU 23: Eliminar todos los recursos seleccionados.

- TI 23: Eliminar todos los recursos seleccionados.

HU 24: Seleccionar todos los servicios.

- TI 24: Seleccionar todos los servicios.

HU 25: Deseleccionar todos los servicios.

- TI 25: Deseleccionar todos los servicios.

HU 26: Eliminar todos los servicios seleccionados.

- TI 26: Eliminar todos los servicios seleccionados.

HU 27: Seleccionar todos los usuarios.

- TI 27: Seleccionar todos los usuarios.

HU 28: Deseleccionar todos los usuarios.

- TI 28: Deseleccionar todos los usuarios.

HU 29: Eliminar todos los usuarios seleccionados.

- TI 29: Eliminar todos los usuarios seleccionados.

Tabla 3.5. Tarea de desarrollo # 5

Tarea	
Número de tarea: 5	**Número de Historia de usuario:** 21
Nombre de la tarea: Seleccionar todos los recursos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 6 de noviembre de 2024	**Fecha de fin:** 6 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite seleccionar a la vez todos los recursos.	

Tabla 3.6. Tarea de desarrollo # 6

Tarea	
Número de tarea: 6	**Número de Historia de usuario:** 22
Nombre de la tarea: Deseleccionar todos los recursos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 7 de noviembre de 2024	**Fecha de fin:** 7 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite deseleccionar a la vez todos los recursos que estaban seleccionados.	

3.2. Fase IV: Pruebas

Las pruebas de software son un proceso fundamental dentro del ciclo de desarrollo de sistemas, cuyo objetivo principal es verificar y validar que el software cumpla con los requisitos especificados y funcione según lo esperado en diversos escenarios. Este proceso implica la ejecución sistemática de una aplicación o sistema con el fin de identificar defectos o desviaciones respecto al comportamiento deseado (Irrazabal y Mascheroni, 2022). Sin embargo, a pesar de su relevancia, las pruebas de software están limitadas en su capacidad para garantizar la ausencia total de errores, una realidad expresada magistralmente por Edsger W. Dijkstra en su afirmación: *"Las pruebas pueden mostrar sólo la presencia de errores, mas no su ausencia"* (Dijkstra; Dahl y Hoare, 1972).

En el contexto de la metodología Programación Extrema (XP), las pruebas ocupan un lugar central como práctica fundamental para garantizar la calidad del software. Este marco propone un enfoque iterativo y altamente colaborativo, donde las pruebas son utilizadas no solo como un medio para validar el funcionamiento del sistema, sino también como una herramienta para guiar el diseño y el desarrollo continuo del mismo. XP sugiere explícitamente la realización de dos tipos esenciales de pruebas: unitarias y de aceptación, cada una de las cuales aborda aspectos específicos del aseguramiento de calidad en el ciclo de vida del software.

3.2.1. Pruebas unitarias

Una prueba unitaria de software, también conocida como *unit testing*, es el instrumento utilizado para validar un fragmento de código fuente. Los desarrolladores aíslan una línea del lenguaje codificado para saber si el sistema está operando correctamente en una función, proceso o actividad específica. La palabra unidad alude a un componente individual del sistema que, a su vez, es desglosado por el programa de *testing* para obtener información detallada sobre el funcionamiento y los comportamientos que lo definen (TestingIt, 2024).

Para la realización de pruebas unitarias en la actual propuesta, se implementaron casos de prueba utilizando Pytest, una herramienta ampliamente reconocida en el ecosistema de Python por su simplicidad y potencia.

Estas fueron diseñadas y ejecutadas en un entorno basado en Django REST Framework, lo que permitió validar la funcionalidad de las API REST desarrolladas en el sistema. A continuación, se presentan algunos de los casos de prueba ejecutados por iteración, los restantes pueden ser consultados en los anexos de la investigación (Ver A.6.1).

Pruebas unitarias para la Iteración 1: Durante esta primera iteración, se llevaron a cabo 6 casos de pruebas unitarias.

```
20      def test_login_successful(self):
21          print('-'*100, '\n', 'testing login successful initiated')
22          data = {
23              'username': 'admin',
24              'password': 'Admin@2024'
25          }
26          response = self.client.post('/api/login/', data, format='json')
27          self.assertEqual(response.status_code, status.HTTP_200_OK)
28          self.assertTrue('token' in response.data)
29          print('login successful test passed')
30
```

Figura 3.1. Caso de prueba unitaria Autenticar usuario.

```
def test_create_resource_successful(self):
    print('-'*100,'\n','testing create resource successful initiated')
    data = {
        'name': 'Test Resource',
        'quantity': 10,
        'price': 15.99
    }
    response = self.client.post(self.url, data)
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertTrue(Resource.objects.filter(name='Test Resource').exists())
    print('create resource successful test passed')
```

Figura 3.2. Caso de prueba unitaria Registar recurso.

Pruebas unitarias para la Iteración 2: Durante esta segunda iteración, se llevaron a cabo 9 casos de pruebas unitarias.

```
    def test_delete_resource_successful(self):
        print('-'*100,'\n','testing delete resource successful initiated')
        data = {'id': self.resource.id}
        response = self.client.post(self.url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        self.assertFalse(Resource.objects.filter(id=self.resource.id).exists())
        print('delete resource successful test passed')
```

Figura 3.3. Caso de prueba unitaria Eliminar recurso.

```
    def test_delete_service(self):
        print('-'*100, '\n', 'testing delete service initiated')
        service = Service.objects.create(
            name='Service to Delete',
            descripcion='Description',
            cost='19.99'
        )
        data = {'id': service.id}
        response = self.client.post('/api/delete_service/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        print('delete service test passed')
```

Figura 3.4. Caso de prueba unitaria Eliminar servicio.

Pruebas unitarias para la Iteración 3: Durante esta tercera iteración, se llevaron a cabo 8 casos de pruebas unitarias.

```
def test_select_all_resources(self):
    print('-'*100, '\n', 'testing select all resources initiated')

    resources = [
        {'id': 1, 'name': 'Resource 1', 'quantity': 10, 'price': '9.99'},
        {'id': 2, 'name': 'Resource 2', 'quantity': 20, 'price': '19.99'}
    ]

    response = self.client.post('/api/resources/', {
        'resources': resources
    }, format='json')

    self.assertEqual(response.status_code, status.HTTP_200_OK)
    print('select all resources test passed')
```

Figura 3.5. Caso de prueba unitaria Seleccionar todos los recursos.

```python
def test_select_all_users(self):

    print('-'*100, '\n', 'testing select all users initiated')

    data1 = {
        'username': 'user1',
        'password': 'Test2024',
        'first_name': 'First',
        'last_name': 'User',
        'role': 'worker'
    }
    data2 = {
        'username': 'user2',
        'password': 'Test2024',
        'first_name': 'Second',
        'last_name': 'User',
        'role': 'admin'
    }

    self.client.post(self.register_url, data1)
    self.client.post(self.register_url, data2)

    response = self.client.get('/api/get_all_users/')

    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(len(response.data['users']), 2)
    print('select all users test passed')
```

Figura 3.6. Caso de prueba unitaria Seleccionar todos los usuarios.

Resultados de las pruebas unitarias.

Se realizaron 23 casos de pruebas unitarias con el propósito de validar la funcionalidad, estabilidad y cumplimiento de los requisitos establecidos para el sistema. En la primera iteración se identificaron 2 defectos y en la segunda iteración se detectó un solo defecto. Tras implementar las correcciones necesarias, se efectuó una tercera iteración en la cual no se identificaron defectos, confirmando que los módulos evaluados operaban conforme a los criterios funcionales establecidos. A continuación, se listan los defectos detectados en cada una de las iteraciones y un gráfico que detalla la relación entre los defectos que fueron detectados y los que fueron resueltos.

Iteración 1

- Error en la validación de credenciales de autenticación: el sistema permitía la autenticación de usuarios con contraseñas incorrectas debido a una validación incompleta en la función login().

- Error en la validación de datos al registrar recursos: el método create_resource() aceptaba números negativos debido a una ausencia de validación en el campo cantidad.

Iteración 2

- Errore de validación en la búsqueda de servicios: el método get_services_with_search() no devolvía todos los servicios coincidentes debido a una validación incompleta en los filtros de búsqueda.

Iteración 3

- No se identificaron nuevos defectos, ya que todos los encontrados en las iteraciones previas fueron corregidos exitosamente.

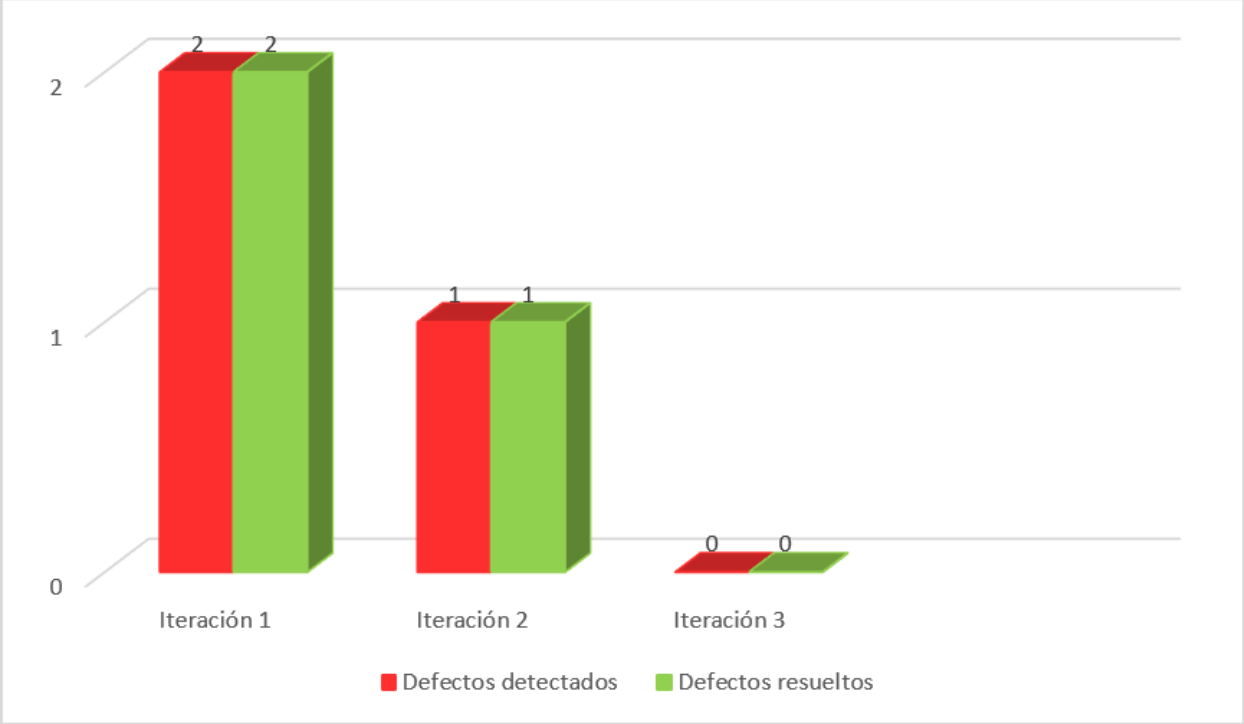


Figura 3.7. Defectos encontrados y defectos resueltos en cada iteración.

3.2.2. Pruebas de aceptación

Las pruebas de aceptación, también llamadas pruebas del cliente, son especificadas por este y se centran en las características y funcionalidades generales del sistema, que son visibles y revisables por parte del clien-

te. Dichas pruebas derivan de las historias de usuario que se han implementado como parte de la liberación del software (Pressman, 2010). A continuación, se presentan algunos de los casos de prueba de aceptación realizados por cada iteración, los restantes pueden ser consultados en los anexos de la investigación (Ver A.7).

Iteración 1: Para la primera iteración, se definió un conjunto de 10 casos de prueba de aceptación (CPA).

Tabla 3.7. Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_CPA1	**Historia de usuario:** 1
Nombre: Autenticar usuario	
Descripción: Prueba para la funcionalidad Autenticar usuario.	
Condiciones de ejecución: • El usuario debe estar previamente registrado en el sistema.	
Pasos de ejecución: • El usuario accede a la página de inicio de sesión del sistema. • El usuario introduce su nombre de usuario y contraseña. • El usuario presiona el botón de Iniciar sesión. • El sistema valida las credenciales y autentica al usuario si son correctas.	
Resultados esperados: • Si las credenciales son correctas, el sistema redirige al usuario a su panel principal según el rol que posea. • Si las credenciales son incorrectas, el sistema muestra un mensaje de error indicando que el usuario o la contraseña son incorrectos.	

Tabla 3.8. Prueba de aceptación # 2

Caso de prueba de aceptación	
Código: HU2_CPA2	**Historia de usuario:** 2
Nombre: Registrar usuario	
Descripción: Prueba para la funcionalidad Registrar usuario.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema.	

Continúa en la próxima página

Tabla 3.8. Continuación de la página anterior

Pasos de ejecución: • El administrador accede a la sección Usuarios. • El administrador selecciona el botón con ícono de adicionar. • Se despliega un formulario con los campos requeridos (nombre, apellidos, nombre de usuario, contraseña y rol). • El administrador llena todos los campos obligatorios con información válida. • El administrador presiona el botón Guardar.
Resultados esperados: • Si todos los datos son válidos, el sistema registra al nuevo usuario y lo incluye en la lista de usuarios visibles. • Si algún campo obligatorio no se llena o contiene datos inválidos, el sistema muestra un mensaje de error indicando el problema.

Iteración 2: Para la segunda iteración, se definió un conjunto de 10 casos de prueba de aceptación (CPA).

Tabla 3.9. Prueba de aceptación # 3

Caso de prueba de aceptación	
Código: HU11_CPA11	**Historia de usuario:** 11
Nombre: Eliminar recurso.	
Descripción: Prueba para la funcionalidad Eliminar recurso.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un recurso registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Recursos. • El administrador localiza el recurso que desea eliminar en la lista de recursos. • El administrador selecciona el ícono de papelera asociado al recurso. • El sistema despliega una ventana emergente solicitando la confirmación de eliminación con el mensaje: ¿Está seguro que desea eliminar este recurso? • El administrador confirma la eliminación presionando el botón Eliminar.	
Resultados esperados: • Si el administrador confirma, el sistema elimina el recurso seleccionado y lo remueve de la lista de recursos. • Si el administrador cancela la acción, el sistema mantiene intacta la información del recurso.	

Tabla 3.10. Prueba de aceptación # 4

Caso de prueba de aceptación	
Código: HU12_CPA12	**Historia de usuario:** 12
Nombre: Listar recursos.	
Descripción: Prueba para la funcionalidad Listar recursos.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un recurso registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Recursos. • El sistema muestra una tabla con las columnas Nombre, Cantidad, Costo y Opciones.	
Resultados esperados: • La lista muestra todos los recursos registrados con la información correspondiente. • Si no hay recursos registrados, se muestra el mensaje de que no hay recursos disponibles.	

Iteración 3: Para la tercera iteración, se definió un conjunto de 9 casos de prueba de aceptación (CPA).

Tabla 3.11. Prueba de aceptación # 5

Caso de prueba de aceptación	
Código: HU21_CPA21	**Historia de usuario:** 21
Nombre: Seleccionar todos los recursos.	
Descripción: Prueba para la funcionalidad Seleccionar todos los recursos.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir recursos listados.	
Pasos de ejecución: • El administrador accede a la sección Recursos. • El administrador presiona el botón verde flotante con el ícono de *check*.	
Resultados esperados: • Todos los recursos de la lista quedan seleccionados. • Si no hay recursos listados, el botón está deshabilitado.	

Tabla 3.12. Prueba de aceptación # 6

Caso de prueba de aceptación	
Código: HU22_CPA22	**Historia de usuario:** 22
Nombre: Deseleccionar todos los recursos.	
Descripción: Prueba para la funcionalidad Deseleccionar todos los recursos.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir recursos seleccionados.	
Pasos de ejecución: • El administrador accede a la sección Recursos. • El administrador presiona el botón gris flotante con el ícono de X.	
Resultados esperados: • Todos los recursos seleccionados quedan desmarcados. • Si no hay recursos seleccionados, el botón está deshabilitado.	

Resultados de las pruebas de aceptación.

Como parte del proceso de evaluación del sistema y con el objetivo principal de identificar la medida en que las funcionalidades implementadas cumplían con los requerimientos establecidos, se llevaron a cabo tres iteraciones de pruebas de aceptación.

Durante la primera iteración, se identificaron 2 no conformidades (NC). Posteriormente, tras la corrección de estas, se procedió a una segunda iteración de pruebas en la que se detectó una nueva no conformidad. Tras implementar las correcciones correspondientes, se realizó una tercera iteración donde no se identificaron nuevas no conformidades. A continuación, se presentan los resultados obtenidos tras la ejecución de las pruebas de aceptación, organizados por iteración.

Iteración 1

- Error ortográfico en la etiqueta del botón de autenticación: el botón para iniciar sesión estaba etiquetado como Iniciar sessión en lugar de Iniciar sesión.
- Error ortográfico en el formulario de registro de recurso: en el formulario de registrar recurso, al seleccionar el botón Cancelar, la pregunta de confirmación no presentaba signos de interrogación.

Iteración 2

- Error ortográfico en el formulario de modificar usuario: en el formulario de modificar usuario, al seleccionar el botón Aceptar, en el mensaje de confirmación estaba escrito Usuario modificado con exito en lugar de Usuario modificado con éxito.

Tabla 3.13. Resumen de las pruebas de aceptación para la Iteración 1

Iteración	Cod. prueba	Resultado
1	HU1_CPA1	No Satisfactorio
1	HU2_CPA2	Satisfactorio
1	HU3_CPA3	No Satisfactorio
1	HU4_CPA4	Satisfactorio
1	HU5_CPA5	Satisfactorio
1	HU6_CPA6	Satisfactorio
1	HU7_CPA7	Satisfactorio
1	HU8_CPA8	Satisfactorio
1	HU9_CPA9	Satisfactorio
1	HU10_CPA10	Satisfactorio

Tabla 3.14. Resumen de las pruebas de aceptación para la Iteración 2

Iteración	Cod. prueba	Resultado
2	HU11_CPA11	Satisfactorio
2	HU12_CPA12	Satisfactorio
2	HU13_CPA13	Satisfactorio
2	HU14_CPA14	Satisfactorio
2	HU15_CPA15	Satisfactorio
2	HU16_CPA16	Satisfactorio
2	HU17_CPA17	Satisfactorio
2	HU18_CPA18	No Satisfactorio
2	HU19_CPA19	Satisfactorio
2	HU20_CPA20	Satisfactorio

Iteración 3

- No se identificaron nuevas no conformidades, ya que todas las encontradas en las iteraciones previas fueron corregidas exitosamente.

3.3. Conclusiones parciales.

Este capítulo ha sido crucial para consolidar el desarrollo del Sistema para la gestión de los recursos y servicios en el taller de impresiones Fotoché:

- El establecimiento de estándares de codificación fue fundamental para garantizar la consistencia, calidad y mantenibilidad del código.

- Las tareas de ingeniería facilitaron la implementación efectiva de las funcionalidades solicitadas, ofreciendo una estructura bien organizada y enfocada en lograr resultados.

- La realización de pruebas unitarias de forma automatizada permitió validar la funcionalidad interna de los módulos del sistema.

- Las pruebas de aceptación confirmaron su alineación con los requerimientos específicos del taller.

Conclusiones

Una vez culminada la investigación, se puede afirmar que se dio cumplimiento a los objetivos planteados, arribando a las siguientes conclusiones:

- El estudio realizado a los sistemas de gestión de recursos y servicios reveló que las soluciones existentes no cubren adecuadamente las necesidades específicas del taller Fotoché, lo que subraya la importancia y la necesidad de esta investigación para mejorar los procesos de gestión de recursos y servicios.

- La identificación de requisitos, junto con el análisis y diseño del sistema, permitió establecer una base sólida para la implementación de la solución adaptada a las necesidades específicas del taller Fotoché.

- Se desarrolló un sistema informático que mejoró la gestión de los recursos y servicios en el taller de impresiones Fotoché, brindando una herramienta eficiente para el control y la organización de los recursos necesarios en las operaciones diarias.

- Las pruebas realizadas y la corrección de las deficiencias detectadas en cada una de las fases de desarrollo del sistema permitieron garantizar un producto final funcional y de calidad, capaz de atender las demandas operativas del taller de manera eficaz.

Recomendaciones

A partir de los resultados obtenidos, se recomienda:

- Desarrollo de un módulo para la emisión de notificaciones: Implementar un módulo que permita enviar notificaciones al administrador cada vez que se requiera el reabastecimiento de recursos en el taller. Este debe ser capaz de identificar las cantidades disponibles y las que se han agotado.

Referencias bibliográficas

ABBA, Ihechikara, 2022. *What is an ORM The Meaning of Object Relational Mapping Database Tools* [online] [visitado 2024-10-22]. Disponible desde: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/ (vid. pág. 27).

ALARCON, Camila, 2024. *Sistema web Â¿Que es?* [Data Trust] [online] [visitado 2024-10-15]. Disponible desde: https://www.datatrust.pe/web/sistema-web/ (vid. pág. 6).

ATLASSIAN, 2024. *What is version control* [online] [visitado 2024-10-26]. Disponible desde: https://www.atlassian.com/git/tutorials/what-is-version-control (vid. pág. 29).

AWATI, Rahul, 2017. *What is computer-aided software engineering? Definition from TechTarget* [ERP] [online] [visitado 2024-10-15]. Disponible desde: https://www.techtarget.com/searcherp/definition/CASE-computer-aided-software-engineering (vid. pág. 18).

BASS, Len; CLEMENTS, Paul y KAZMAN, Rick, 2012. *Software Architecture in Practice, Third Edition, Addison-Wesley* [online] [visitado 2024-10-25]. ISBN 978-0-13-294279-9. Disponible desde: https://learning.oreilly.com/library/view/software-architecture-in/9780132942799/ch01.html (vid. pág. 44).

BECK, Kent y ANDRES, Cynthia, 2004. *Extreme Programming Explained: Embrace Change, Second Edition, Addison-Wesley, 2004.* [online] [visitado 2024-10-15]. ISBN 978-0-321-27865-4. Disponible desde: https://learning.oreilly.com/library/view/extreme-programming-explained/0321278658/ch01.html (vid. págs. 16-18).

BLANCARTE, Oscar Javier, 2016. *Introduccion a los patrones de Disenno-Facade Patron de disenno estructural* [online] [visitado 2024-10-27]. Disponible desde: https://reactiveprogramming.io/blog/es/patrones-de-diseno/facade (vid. pág. 57).

BRUTTI, Franco, 2023. *UML: el lenguaje universal para el modelado de sistemas*. Url: https://thepower.education/blog/uml-el-lenguaje-universal-para-el-modelado-de-sistemas (vid. págs. 19, 20).

CANELO, Miriam Martinez, 2020. *Que son los Patrones de Disenno de software* [online] [visitado 2024-10-27]. Disponible desde: https://profile.es/blog/patrones-de-diseno-de-software/ (vid. pág. 49).

CARDENAS, Hector, 2023. *Â¿Cuales son los recursos de una empresa? Tipos y ejemplos* [Conekta] [online] [visitado 2024-11-24]. Disponible desde: https://www.conekta.com/blog/recursos-de-una-empresa (vid. págs. 7, 8).

CHAPAVAL, Nicole, 2017. *Que es Frontend y Backend: caracteristicas, diferencias y ejemplos* [online] [visitado 2024-10-21]. Disponible desde: https://platzi.com/blog/que-es-frontend-y-backend/ (vid. pág. 22).

CHAVERRA, Jonathan Antonio Hoyos y ARIAS, Alejandro Valencia, 2012. EL PAPEL DE LAS TIC EN EL ENTORNO ORGANIZACIONAL DE LAS PYMES. Url: https://www.redalyc.org/pdf/5343/534366877001.pdf#page=7.71 (vid. pág. 1).

CHEN, Michael, 2024. *Big Data, Big Possibilities: How to Extract Maximum Value* [online] [visitado 2024-10-22]. Disponible desde: https://www.oracle.com/big-data/what-is-big-data/ (vid. pág. 23).

CIMAS CUADRADO, Gustavo, 2024. *Visual Studio Code: Editor de codigo para desarrolladores* [online] [visitado 2024-10-15]. Disponible desde: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/ (vid. pág. 27).

CLAVIJO, Camilo, 2024. *QuÃ© es un software CRM, para quÃ© sirve y caracterÃsticas* [online] [visitado 2024-10-21]. Disponible desde: https://blog.hubspot.es/sales/que-es-un-software-crm (vid. pág. 10).

CLOCKIFY, 2023. *Gestion de recursos: tipos, etapas y consejos de optimizacion* [Centro de conocimiento de Clockify] [online] [visitado 2024-11-24]. Disponible desde: https://clockify.me/learn/es/business-management/what-is-resource-management/ Section: Freelancers & Contractors (vid. pág. 12).

CREATELY, 2024. *Tarjetas CRC* [online] [visitado 2024-10-30]. Disponible desde: https://creately.com/diagram/example/XvuQx6d1RXf/tarjetas-crc (vid. pág. 44).

CRISTANCHO, Felipe, 2022. *Python: Ventajas y desventajas - Talently* [online] [visitado 2024-10-17]. Disponible desde: https://talently.tech/blog/python-ventajas-y-desventajas/ Section: Lenguajes de Programacion (vid. pág. 23).

DARIAS PEREZ, Sergio, 2021. *Gestor de Base de datos: Que es, Funcionalidades y Ejemplos* [online] [visitado 2024-10-17]. Disponible desde: https://intelequia.com/es/blog/post/gestor-de-base-de-datos-qu%C3%83%C2%A9-es-funcionalidades-y-ejemplos (vid. pág. 28).

DATAZUCAR, 2024. *Versat Sarasola* [online] [visitado 2024-12-01]. Disponible desde: https://www.datazucar.cu/?featured_item=another-print-package (vid. pág. 11).

DIJKSTRA, E. W.; DAHL, 0.-J. y HOARE, C. A. R., 1972. *Structured Programming. Londres: Academic Press.* (Vid. pág. 68).

DOMAIN LOGIC, 2022. *Metodologias de desarrollo de software 2022* [Domain Logic] [online] [visitado 2024-11-24]. Disponible desde: https://domainlogic.io/metodologias-de-desarrollo-de-software-2022/ (vid. pág. 14).

DORANTES, Cesar Anton, 2015. *Descubre PostgreSQL: que es, como funciona y ventajas* [online] [visitado 2024-10-15]. Disponible desde: https://platzi.com/blog/que-es-postgresql/ (vid. pág. 28).

EKON, Equipo, 2021. *Â¿Que es un sistema de gestion y para que sirve?* [Ekon] [online] [visitado 2024-10-24]. Disponible desde: https://www.ekon.es/blog/sistemas-de-gestion-integral-para-el-funcionamiento-optimo-de-la-empresa/ (vid. págs. 5, 6).

ESPACIOERP, 2024. *ERPNext: Resenna | Conoce los modulos de esta solucion* [online] [visitado 2024-11-16]. Disponible desde: https://espacioerp.com/resenas-erpnext/ (vid. pág. 10).

FASTERCAPITAL, 2024. *Planificacion e Iteraciones en Programacion Extrema* [online] [visitado 2024-10-30]. Disponible desde: https://fastercapital.com/keyword/planificaci%C3%83%C2%B3n-e-iteraciones-en-programaci%C3%83%C2%B3n-extrema.html (vid. pág. 41).

FERNANDEZ, Yubal, 2019. *API: que es y para que sirve* [online] [visitado 2024-10-26]. Disponible desde: https://www.xataka.com/basics/api-que-sirve (vid. pág. 46).

FOUNDATION, Django Software, 2024. *The web framework for perfectionists with deadlines* [online] [visitado 2024-10-15]. Disponible desde: https://www.djangoproject.com/ (vid. pág. 27).

FOUNDATION, Dolibarr, 2023. *Dolibarr Open Source ERP y CRM: paquete web para empresas* [online] [visitado 2024-11-16]. Disponible desde: https://www.dolibarr.org/dolibarr-home.php (vid. págs. 10, 11).

FOWLER, Martin y SCOTT, Kendall, 1999. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Second Edition*. ISBN 978-0-201-65783-8. Url: https://learning.oreilly.com/library/view/uml-distilled-a/020165783X/020165783X_ch11.html (vid. pág. 19).

GACKENHEIMER, Cory, 2015. *Introduction to React*. Apress. ISBN 978-1-4842-1245-5. Url: https://books.google.com/books?hl=es&lr=&id=NZCKCgAAQBAJ&oi=fnd&pg=PR6&dq=react&ots=KCuqWnHC6h&sig=Z7j8dOcsPt0BM26N4BtrTipUzWw#v=onepage&q=react&f=false (vid. págs. 25, 26).

GALICIA, Esteban, 2019. *Patron de disenno: Cadena de responsabilidad* [online] [visitado 2024-10-27]. Disponible desde: https://blog.thedojo.mx/2019/02/21/cadena-responsabilidades.html (vid. pág. 58).

GAUCHAT, Juan Diego, 2012. *El gran libro de HTML5, CSS3 y Javascript*. Marcombo. ISBN 978-84-267-1782-5. Url: https://books.google.com/books?hl=es&lr=&id=szDMlRzwzuUC&oi=fnd&pg=PA1&dq=html5&ots=0DoFYZwASf&sig=VwXeJNDefbnE22z9Y0VIp-XRbpY#v=onepage&q=html5&f=false (vid. págs. 23, 24).

GEOGRAPHIC, National, 2017. *La revolucion digital que esta transformando Cuba* [National Geographic] [online] [visitado 2024-11-15]. Disponible desde: https://www.nationalgeographic.es/viaje-y-aventuras/2017/05/la-revolucion-digital-que-esta-transformando-cuba Section: Viaje y Aventuras (vid. pág. 1).

GOLDBERG, Josh, 2022. *Learning TypeScript*. O'Reilly Media, Inc. ISBN 978-1-0981-1030-7 (vid. pág. 22).

GORKOVENKO, Andrey, 2023. *Patrones de disenno: metodo de fabrica* [CodeGym] [online] [visitado 2024-10-27]. Disponible desde: https://codegym.cc/es/groups/posts/es.285.patrones-de-diseno-metodo-de-fabrica (vid. pág. 55).

GROUP, The PostgreSQL Global Development, 2024. *PostgreSQL* [online] [visitado 2024-10-16]. Disponible desde: https://www.postgresql.org/ (vid. pág. 28).

GUERRERO, Carlos A.; SUAREZ, Johanna M. y GUTIERREZ, Luz E., 2013. Patrones de Disenno GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web [online] [visitado 2024-11-15]. ISSN 0718-0764. Disponible desde: http://www.scielo.cl/scielo.php?script=sci_abstract&pid=S0718-07642013000300012&lng=es&nrm=iso&tlng=en (vid. pág. 54).

HALLIDAY, Paul y GORTON, Cristina, 2021. *How To Use Axios with React* [online] [visitado 2024-11-20]. Disponible desde: https://www.digitalocean.com/community/tutorials/react-axios-react (vid. pág. 47).

IDEAFOSTER, 2023. *Herramientas de prototipado: guia completa para elegir la adecuada* [online] [visitado 2024-10-15]. Disponible desde: https://ideafoster.com/es/herramientas-prototipado/ (vid. pág. 20).

IONOS, 2020. *El patron Composite: ejemplos de soluciones para jerarquias parte-todo* [online] [visitado 2024-10-27]. Disponible desde: https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/patron-composite/ (vid. pág. 54).

IRRAZABAL, Emanuel y MASCHERONI, Maximiliano, 2022. *Fundamentos de las pruebas continuas de software (Primera Edicion)*. Natalia Passicot. Editorial de la Universidad Nacional del Nordeste (vid. pág. 68).

JUMMP, 2011. *Desarrollo de software. Programacion extrema II* [online] [visitado 2024-11-16]. Disponible desde: https://jummp.wordpress.com/2011/04/27/desarrollo-de-software-programacion-extrema-extreme-programming-xp-ii/ (vid. pág. 63).

KEHR, Paula, 2024. *Â¿Que son los puntos de historia? Definicion y ejemplos* [online] [visitado 2024-10-29]. Disponible desde: https://www.instagantt.com/es/gestion-de-proyectos/que-son-los-puntos-de-historia (vid. pág. 33).

KEN, Arizbe, 2023. *Arquitectura de software: Â¿Que es y que tipos hay?* [online] [visitado 2024-10-25]. Disponible desde: https://gluo.mx/blog/arquitectura-de-software-que-es-y-que-tipos-hay (vid. pág. 45).

KOTHARI, Amit, 2024. *Todo lo que necesita saber sobre los diagramas UML* [online] [visitado 2024-10-21]. Disponible desde: https://tallyfy.com/uml-diagram/ (vid. pág. 19).

KOTLER, Philip y ARMSTRONG, Gary, 1997. *Marketing: An Introduction. Sexta Edicion.* [online] [visitado 2024-11-24]. Disponible desde: https://books.google.ca/books?hl=es&lr=&id=sLJXV_z8XC4C&oi=fnd&pg=PA9&dq=Kotler,+P.+(1997).+Mercadotecnia.+M%C3%A9xico:+Prentice-Hall&ots=IgJi5bD5Pn&sig=Izl48DNXJJtd3qgHPP8X8KmST6M#v=onepage&q=Kotler%2C%20P.%20(1997).%20Mercadotecnia.%20M%C3%A9xico%3A%20Prentice-Hall&f=false (vid. pág. 8).

LIZAMA, Oscar; KINDLEY, Geordy; MORALES, Javier Ignacio Jeria y GONZALES, Agustin, 2016. Redes de computadores Arquitectura Cliente - Servidor (vid. pág. 45).

LUCENA, Paola, 2023. *Â¿Que es el framework?* Url: https://www.cesuma.mx/blog/que-es-el-framework.html (vid. pág. 25).

MACIAS, Maria Auxiliadora y GIL, Darwin, 2018. LA DEMANDA Y SU INFLUENCIA EN LA CAPACIDAD DE LA ORGANIZACION. *Contribuciones a la Economia*. Url: https://www.eumed.net/rev/ce/2018/2/demanda-organizacion.html (vid. pág. 1).

MARTINS, Julia, 2024. *Â¿Que es la metodologia Kanban y como funciona?* [Asana] [online] [visitado 2024-11-25]. Disponible desde: https://asana.com/es/resources/what-is-kanban (vid. pág. 15).

MATCZAK, Marta, 2023. *Who is Guido van Rossum? His major achievements* [online] [visitado 2024-10-21]. Disponible desde: https://nofluffjobs.com/log/off-duty-en/who-is-guido-van-rossum-his-major-achievements/ (vid. pág. 22).

MEYERDELIUS, Harald, 2022. *Que es un ERP Caracteristicas, Funciones y Beneficios*. Url: https://www.holded.com/es/blog/que-es-erp-y-para-que-sirve Section: Facturacion (vid. pág. 9).

MIGALLON, Laura, 2021. *Que tipos de servicios existen y su definicion* [Witei] [online] [visitado 2024-11-24]. Disponible desde: https://get.witei.com/es/articulos/que-tipos-de-servicios-existen-y-su-definicion/ (vid. pág. 8).

MORA, Roberto Canales, 2003. *Patrones GRASP*. Url: https://adictosaltrabajo.com/2003/12/22/grasp/ (vid. pág. 50).

MORO RAMOS, andra, 2024. *Canva como herramienta para promover el aprendizaje significativo en la ensennanza del ingles como lengua extranjera.* [online] [visitado 2024-11-16]. Disponible desde: https://epsir.net/index.php/epsir/article/view/869 (vid. págs. 20, 21).

MURILLO, Raquel, 2023. *Patrones de arquitectura de software* [online] [visitado 2024-10-26]. Disponible desde: https://yapiko.com/es/blog/patrones-arquitectura-software/ (vid. pág. 47).

NAVAS LOPEZ, Jose Emilio y GUERRAS MARTIN, Luis Angel, 2007. *La direccion estrategica de la empresa: teoria y aplicaciones* [online]. Thomson Reuters-Civitas [visitado 2024-11-24]. ISBN 978-84-470-2850-4. Disponible desde: https://dialnet.unirioja.es/servlet/libro?codigo=138115 (vid. pág. 7).

NEKRASOV, Hlib A., 2023. Coding Standards and Quality Checks: Key Factors for Successful Software Development. En: *Coding Standards and Quality Checks: Key Factors for Successful Software Development* [online] [visitado 2024-11-15]. Disponible desde: https://ieeexplore.ieee.org/document/10346356 (vid. pág. 61).

NUNNEZ, Eduardo, 2022. *Curva de aprendizaje: Â¿que es y para que sirve?* [https://www.crehana.com] [online] [visitado 2024-11-29]. Disponible desde: https://www.crehana.com/blog/desempeno/curva-de-aprendizaje/ (vid. pág. 13).

OBANDO, Rafael, 2024. *Control interno empresarial: elementos y objetivos* [online] [visitado 2024-10-24]. Disponible desde: https://blog.hubspot.es/marketing/control-interno#que-es (vid. pág. 7).

PARADIGM, Visual, 2024. *Paradigma Visual - UML, Agile, PMBOK, TOGAF, BPMN* [online] [visitado 2024-10-15]. Disponible desde: https://www.visual-paradigm.com/features/ (vid. pág. 19).

PRESSMAN, Roger S, 2010. *Ingenieria del Software: Un Enfoque Practico (Septima edicion)* (vid. págs. 17, 18, 38, 43, 58, 61, 73).

QUESADA, Alina Karla y MEDINA, Alberto, 2020. *METODOS TEORICOS DE INVESTIGACION: ANALISIS SINTESIS, INDUCCION DEDUCCION, ABSTRACTO CONCRETO E HISTORICO LOGICO*. Url: https://www.researchgate.net/publication/347987929_METODOS_TEORICOS_DE_INVESTIGACION_ANALISIS-SINTESIS_INDUCCION-DEDUCCION_ABSTRACTO_-CONCRETO_E_HISTORICO-LOGICO (vid. págs. 2, 3).

RAMIREZ, Omar, 2024. *Â¿Que es Typescript por que debes aprenderlo?* [online] [visitado 2024-10-15]. Disponible desde: https://bambu-mobile.com/typescript-que-es/ (vid. pág. 21).

RASCIA, Tania, 2023. *Understanding the DOMDocument Object Model*. Url: https://www.digitalocean.com/community/tutorial-series/understanding-the-dom-document-object-model (vid. pág. 26).

RAUTENSTRAUCH, Ramon, 2024. *Resumen de la historia de Microsoft. 1975 a 2024.* [Consultor 365] [online] [visitado 2024-11-16]. Disponible desde: https://www.consultor365.com/ms/historia-microsoft-1975a2024/ (vid. pág. 21).

RECOM+, 2024. *RECOM+ Software para la gestion comercial e inteligencia de negocios*. Url: https://www.eti.cu/es/servicios/recom-software-para-la-gestion-comercial-e-inteligencia-de-negocios (vid. pág. 12).

RICHARDS, Mark y FORD, Neal, 2020. *Fundamentals of Software Architecture, Addison-Wesley, 2020* [online] [visitado 2024-10-25]. ISBN 978-1-4920-4344-7. Disponible desde: https://learning.oreilly.com/library/view/fundamentals-of-software/9781492043447/ch01.html (vid. pág. 45).

ROOTSTACK, 2024. *Let's talk about ERPNext | Rootstack* [online] [visitado 2024-11-16]. Disponible desde: https://rootstack.com/en/blog/lets-talk-about-erpnext (vid. pág. 10).

ROUSE, Margaret, 2024. *Intellisense* [online] [visitado 2024-10-22]. Disponible desde: https://www.techopedia.com/es/definicion/intellisense (vid. pág. 27).

RUBIO, Daniel, 2017. Introduction to the Django Framework. En: *Introduction to the Django Framework* [online]. Apress [visitado 2024-11-15]. ISBN 978-1-4842-2787-9. Disponible desde: https://doi.org/10.1007/978-1-4842-2787-9_1 (vid. pág. 26).

SANTANDER OPEN ACADEMY, 2024. *Metodologias de desarrollo de software: Â¿que son?* [online] [visitado 2024-10-14]. Disponible desde: https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html (vid. págs. 13, 14).

SCHULZ, Ralph G., 2009. *Disenno web con CSS*. ISBN 978-84-267-1470-1. Url: https://books.google.com/books?hl=es&lr=&id=ZgI2WfHjPiEC&oi=fnd&pg=PA35&dq=CSS&ots=7pRqhVZO-g&sig=DOe2Y6PzCbCdtmlBtxifZqU_q5E#v=onepage&q=CSS&f=false (vid. págs. 24, 25).

SCHWABER, Ken y SUTHERLAND, Jeff, 2022. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game* [online] [visitado 2024-11-25]. Disponible desde: https://www.bing.com/search?q=gu%c3%ada+scrum+2022&FORM=QSRE1 (vid. pág. 15).

SEBESTA, Robert W, 2015. Concepts of Programming Languages, Eleventh Edition, Global Edition (vid. pág. 21).

SISTERNAS, Pau, 2024. *Ejemplos de necesidades y expectativas del cliente* [online] [visitado 2024-11-25]. Disponible desde: https://emprendepyme.net/ejemplos-de-necesidades-de-los-clientes.html (vid. pág. 1).

SOMMERVILLE, Ian, 2011. Ingenieria de Software (Novena Edicion) (vid. págs. 32, 37, 47).

SOUZA, 2019. *Â¿QuÃ© es XML y para quÃ© sirve este lenguaje de marcado?* [online] [visitado 2024-10-23]. Disponible desde: https://rockcontent.com/es/blog/que-es-xml/ (vid. pág. 28).

STACKSCALE, 2021. *What is open source?* [Stackscale Grupo Aire] [online] [visitado 2024-11-24]. Disponible desde: https://www.stackscale.com/blog/what-is-open-source/ (vid. pág. 12).

STRAUB, Ben y CHACON, Scott, 2014. *Git* [online] [visitado 2024-10-15]. Disponible desde: https://git-scm.com/book/en/v2 (vid. págs. 27, 29).

SUFIYAN, Taha, 2024. *An Introduction to JavaScript: Here Is All You Need to Know* [online] [visitado 2024-10-21]. Disponible desde: https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript (vid. pág. 21).

TABARES, Ricardo Botero, 2010. *Patrones Grasp y Anti-Patrones: un Enfoque Orientado a Objetos desde Logica de Programacion*. Url: https://revistas.ucp.edu.co/index.php/entrecienciaeingenieria/article/download/753/742 (vid. págs. 50-52).

TAGLIAFERRI, Lisa, 2022. *An Introduction to JSON* [online] [visitado 2024-10-23]. Disponible desde: https://www.digitalocean.com/community/tutorials/an-introduction-to-json (vid. pág. 28).

TECHNOLOGIES, O2b, 2023. *Odoo Community vs Enterprise: Key Differences Explained* [O2b Technologies] [online] [visitado 2024-11-13]. Disponible desde: https://www.o2btechnologies.com/blog/odoo/odoo-community-vs-enterprise-key-differences-explained (vid. pág. 9).

TENEA, 2024. *Â¿Que es middleware?* [online] [visitado 2024-10-31]. Disponible desde: https://www.tenea.com/tecnologia/servicios-middleware-management (vid. pág. 58).

TESTINGIT, 2024. *Â¿Que son las pruebas unitarias de software?* [online] [visitado 2024-11-17]. Disponible desde: https://www.testingit.com.mx/blog/pruebas-unitarias-de-software (vid. pág. 68).

TIPALTI, 2023. *What is Odoo ERP Software? - System Integration Guide | Tipalti* [online] [visitado 2024-10-14]. Disponible desde: https://tipalti.com/erp-integrations/odoo-erp/ (vid. pág. 9).

TORO BONILLA, Miguel, 2022. *Fundamentos de programacion: PYTHON* [online]. 2022.ª ed. Editorial Universidad de Sevilla [visitado 2024-11-15]. Disponible desde DOI: 10.12795/9788447223602 (vid. pág. 22).

TRIFORK, 2024. *Kent Beck at Copenhagen* [online] [visitado 2024-10-21]. Disponible desde: https://gotocph.com/2024/speakers/3628/kent-beck (vid. pág. 17).

VARGAS, Diego, 2022. *Â¿Que es un entorno de desarrollo y en que se diferencia de un entorno de desarrollo integrado?* [online] [visitado 2024-10-15]. Disponible desde: https://www.hostinger.es/tutoriales/que-es-un-entorno-de-desarrollo (vid. pág. 27).

WELLS, Don, 1999. *User Stories*. Url: http://www.extremeprogramming.org/rules/userstories.html (vid. págs. 38, 40).

YUSUF, Bulent y BRYAN, Alvin, 2023. *What is an API endpoint?* [online] [visitado 2024-10-31]. Disponible desde: https://www.contentful.com/blog/api-endpoint/ (vid. pág. 51).

ZENDESK, 2023. *Â¿Que es gestion de servicios? Guia practica, ventajas y tips* [online] [visitado 2024-11-25]. Disponible desde: https://www.zendesk.com.mx/blog/gestion-de-servicios/ (vid. pág. 12).

Generado con LaTeX: 5 de diciembre de 2024: 11:11am

Apéndices

APÉNDICE A

Apéndices

A.1. Entrevista

Entrevista realizada a: Violeta González Sarquíz, Administradora del taller de impresiones Fotoché.
Fecha: 20/09/2024

Objetivo: Comprender el funcionamiento del taller de impresiones Fotoché, en cuanto a los servicios que allí se brindan y los recursos empleados para la prestación de estos servicios.

1. ¿Podría describir el flujo de los procesos relacionados con la gestión de los recursos y los servicios, desde la solicitud de un servicio por parte del cliente hasta la utilización de los recursos necesarios para completarlo?

2. En caso de inconsistencias, como la falta de recursos para realizar un servicio o errores en el registro de servicios ofrecidos, ¿qué procedimientos se siguen para detectar y resolver estas situaciones?

3. ¿Cuáles son los pasos específicos para realizar ajustes en los registros de recursos y servicios utilizados? ¿Cómo se documentan estos cambios?

4. ¿Qué medidas se toman para garantizar que los recursos estén disponibles en el momento adecuado y evitar retrasos o errores en la prestación de servicios?

5. ¿Qué herramientas o software utilizan actualmente para gestionar tanto los recursos como los servicios? ¿Considera que cumplen con las necesidades del taller?

6. Desde su experiencia, ¿cuáles son las principales normas o directrices que rigen el uso de recursos y la prestación de servicios en el taller?

7. ¿Cuáles son las principales deficiencias que ha identificado en el sistema actual de gestión de recursos y servicios?

8. ¿Qué cambios o mejoras sugeriría para optimizar la planificación y el control de los recursos, asegurando al mismo tiempo la calidad y la eficiencia en la prestación de servicios?

A.2. Encuesta

1. **Información general:** ¿Cuál es su rol en el taller? (Opciones: Trabajador, Administrador)

2. **Funcionalidades deseadas:** ¿Qué características considera más importantes en un sistema de gestión de recursos y servicios? (Seleccione todas las que apliquen):
   - Monitorear la disponibilidad de recursos.
   - Registrar servicios realizados.
   - Calcular ingresos diarios y costos asociados.
   - Otro (especificar): _____
3. **Frecuencia de problemas:** (Opciones: Nunca, Rara vez, A veces, Frecuentemente, Siempre)

4. **Planificación de recursos y servicios:** ¿Qué días considera que requieren mayor planificación de recursos y servicios debido a la alta demanda? (Opciones: Lunes, Martes, Miércoles, Jueves, Viernes)

5. **Impacto en el rendimiento:** ¿Cree que un sistema de gestión integral podría mejorar la eficiencia en la asignación de recursos y servicios? (Opciones: Sí, No, No estoy seguro)

6. **Identificación de problemas y oportunidades**: ¿Considera que un sistema podría ayudar a identificar servicios que generan mayores ingresos o consumen más recursos? (Opciones: Sí, No, No estoy seguro)

7. **Satisfacción del cliente:** En una escala del 1 al 5, ¿cómo calificaría el impacto actual de la gestión de recursos y servicios en la satisfacción del cliente? (1 = Muy bajo, 5 = Muy alto)

A.3. Historias de usuario

Tabla A.1. Historia de usuario # 3

Historia de usuario	
Número: 3	**Nombre:** Registrar recurso
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador registrar un nuevo recurso, llenando los campos Nombre, Cantidad y Costo.	
Observaciones: El administrador deberá estar autenticado en el sistema. Los campos Nombre, Cantidad y Costo son obligatorios.	
Interfaz: 	

Tabla A.2. Historia de usuario # 4

Historia de usuario	
Número: 4	**Nombre:** Modificar recurso
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador modificar un recurso ya registrado, llenando los campos Nombre, Cantidad y Costo.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un recurso creado en el sistema.	

Continúa en la próxima página

Tabla A.2. Continuación de la página anterior

Interfaz:

Tabla A.3. Historia de usuario # 5

Historia de usuario	
Número: 5	**Nombre:** Registrar servicio
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador registrar un nuevo servicio, llenando los campos Nombre, Descripción, Precio, Recursos que consume y Estado.	
Observaciones: El administrador deberá estar autenticado en el sistema. Los campos Nombre, Descripción, Precio y Estado. Es obligatoria la selección de uno o más recursos a consumir.	

Continúa en la próxima página

Tabla A.3. Continuación de la página anterior

Interfaz:

Tabla A.4. Historia de usuario # 6

Historia de usuario	
Número: 6	**Nombre:** Modificar servicio
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador modificar un servicio ya registrado, llenando los campos Nombre, Descripción, Precio, Recursos que consume y Estado.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un servicio creado en el sistema.	

Continúa en la próxima página

Tabla A.4. Continuación de la página anterior

Interfaz:

Tabla A.5. Historia de usuario # 7

Historia de usuario	
Número: 7	**Nombre:** Ofrecer servicio
Usuario: Trabajador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Dará la opción al trabajador de registrar la prestación de un servicio a un cliente, llenando los campos Tipo de servicio y Cantidad.	
Observaciones: El servicio deberá estar previamente habilitado por el administrador para poder ofrecerse.	

Continúa en la próxima página

Tabla A.5. Continuación de la página anterior

Interfaz:

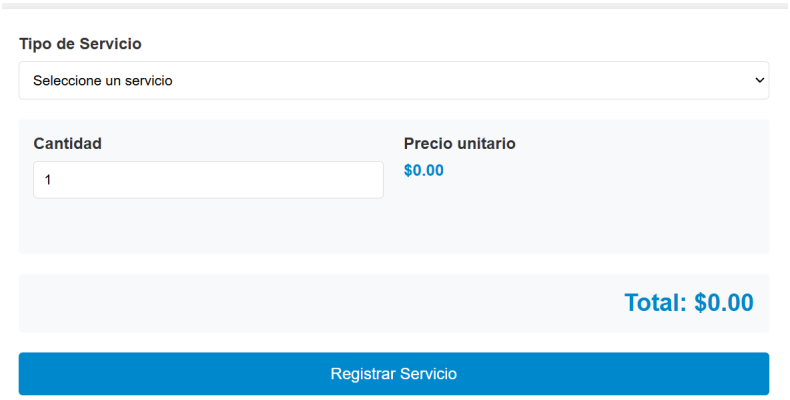


Tabla A.6. Historia de usuario # 8

Historia de usuario	
Número: 8	**Nombre:** Visualizar ganancia total en una fecha determinada
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Ofrecerá al administrador la opción de filtrar por fecha para obtener las ganancias generadas por la prestación de servicios en una fecha determinada.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un servicio ofrecido en la fecha seleccionada que genere ganancias.	

Continúa en la próxima página

Tabla A.6. Continuación de la página anterior

Interfaz:

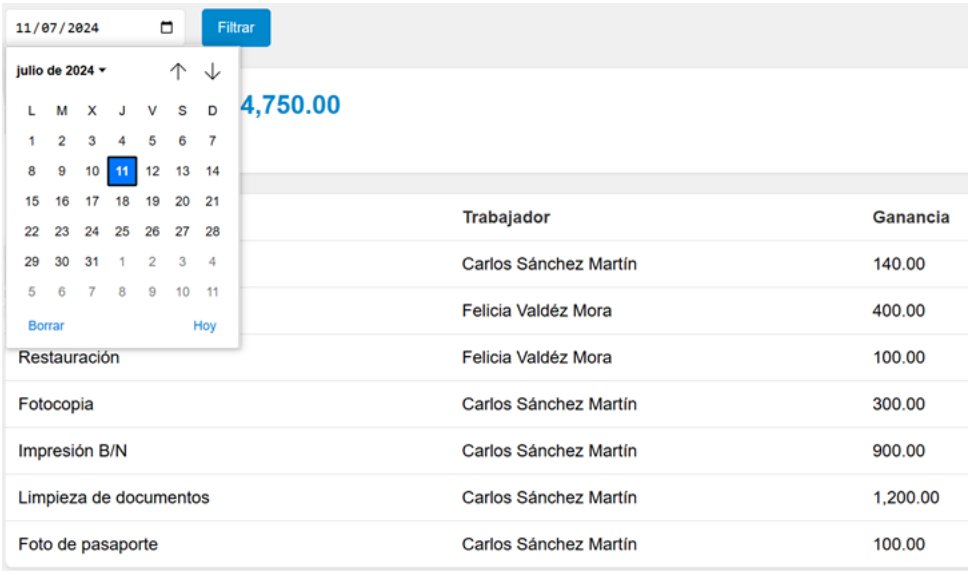


Tabla A.7. Historia de usuario # 9

Historia de usuario	
Número: 9	**Nombre:** Visualizar gasto total de recursos en una fecha determinada
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Ofrecerá al administrador la opción de filtrar por fecha para obtener los gastos derivados del uso de recursos en una fecha determinada. Deberá existir al menos un servicio ofrecido en la fecha seleccionada que genere gasto de recursos.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos el gasto de un recurso en la fecha seleccionada.	

Continúa en la próxima página

Tabla A.7. Continuación de la página anterior

Interfaz:

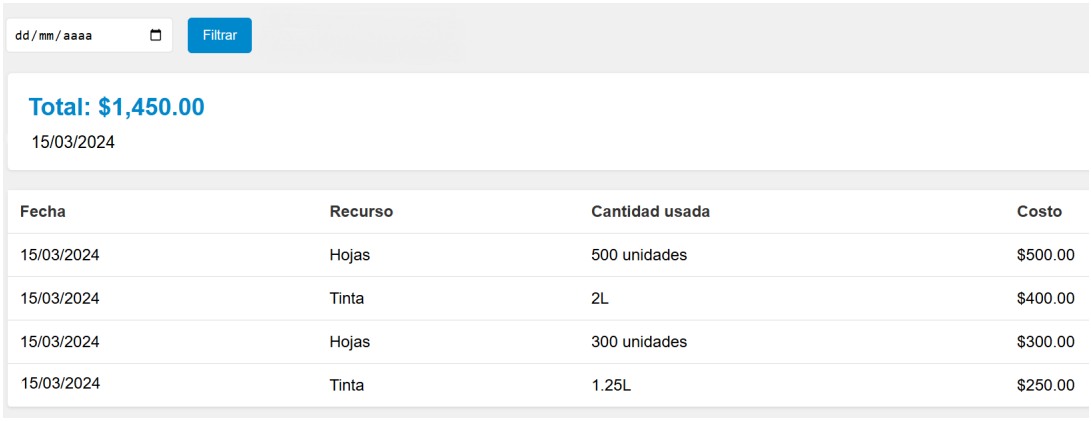


Tabla A.8. Historia de usuario # 10

Historia de usuario	
Número: 10	**Nombre:** Visualizar servicios ofrecidos en una fecha determinada
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Stefanni de la Caridad López González	
Descripción: Ofrecerá al administrador un listado de los servicios ofrecidos en una fecha determinada.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un servicio ofrecido en la fecha seleccionada.	
Interfaz:	

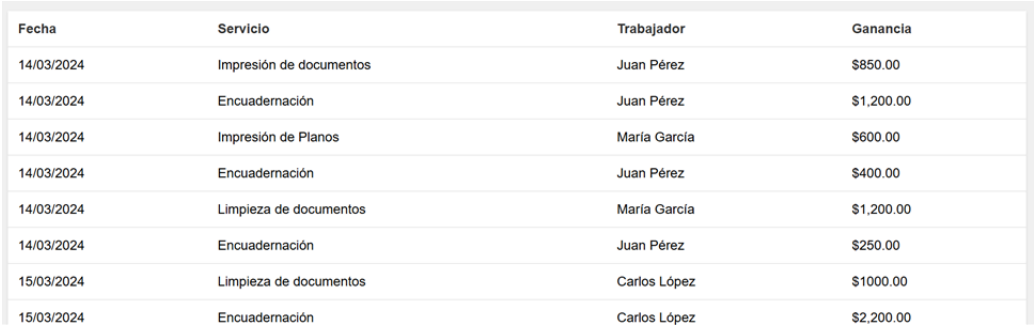


Tabla A.9. Historia de usuario # 11

Historia de usuario	
Número: 11	**Nombre:** Eliminar recurso
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador eliminar un recurso.	
Observaciones: Deberá aparecer una confirmación antes de eliminar.	
Interfaz: 	

Tabla A.10. Historia de usuario # 12

Historia de usuario	
Número: 12	**Nombre:** Listar recursos
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador visualizar todos los recursos en forma de tabla con las columnas Nombre, Cantidad, Costo y Opciones.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un recurso creado en el sistema.	

Continúa en la próxima página

Tabla A.10. Continuación de la página anterior

Interfaz:

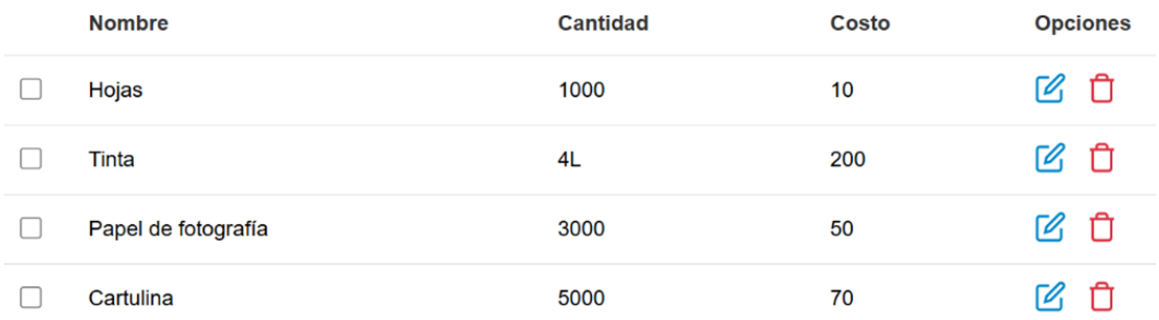


Tabla A.11. Historia de usuario # 13

Historia de usuario	
Número: 13	**Nombre:** Buscar recursos
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador realizar una búsqueda, introduciendo el nombre del recurso, y devolverá una lista de los que coincidan con el criterio de búsqueda.	
Observaciones: El administrador deberá estar autenticado en el sistema.	
Interfaz: 	

Tabla A.12. Historia de usuario # 14

Historia de usuario	
Número: 14	**Nombre:** Eliminar servicio
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador eliminar un servicio.	

Continúa en la próxima página

Tabla A.12. Continuación de la página anterior

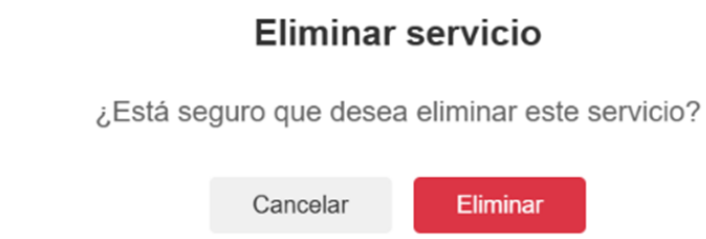
Observaciones: Deberá aparecer una confirmación antes de eliminar.
Interfaz: 

Tabla A.13. Historia de usuario # 15

Historia de usuario	
Número: 15	**Nombre:** Listar servicios
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador visualizar todos los servicios en forma de tabla con las columnas Nombre, Precio, Estado y Opciones.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un servicio creado en el sistema.	
Interfaz: 	

Tabla A.14. Historia de usuario # 16

Historia de usuario

Continúa en la próxima página

Tabla A.14. Continuación de la página anterior

Número: 16	**Nombre:** Buscar servicios
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador realizar una búsqueda, introduciendo el nombre del servicio, y devolverá una lista de los que coincidan con el criterio de búsqueda.	
Observaciones: El administrador deberá estar autenticado en el sistema.	
Interfaz: 	

Tabla A.15. Historia de usuario # 17

Historia de usuario	
Número: 17	**Nombre:** Cerrar sesión
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Ofrecerá al usuario la opción de finalizar su actividad en el sistema de forma segura.	
Observaciones: El usuario deberá estar autenticado en el sistema. La opción de cerrar sesión deberá ser accesible en cualquier momento del uso del sistema.	
Interfaz: 	

Tabla A.16. Historia de usuario # 18

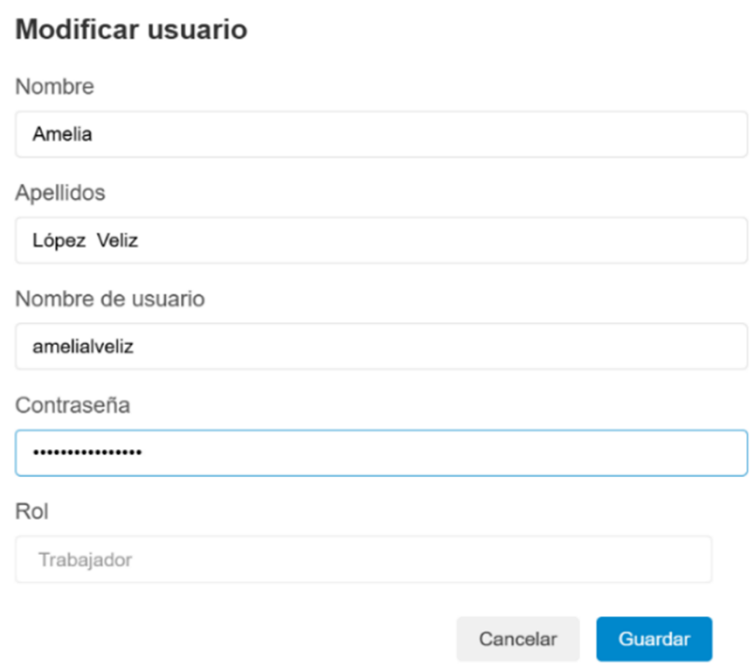
Historia de usuario	
Número: 18	**Nombre:** Modificar usuario
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador modificar un usuario ya registrado, llenando los campos Nombre, Apellidos, Nombre de usuario y Contraseña.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un usuario creado en el sistema.	
Interfaz: 	

Tabla A.17. Historia de usuario # 19

Historia de usuario	
Número: 19	**Nombre:** Eliminar usuario
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	

Continúa en la próxima página

Tabla A.17. Continuación de la página anterior

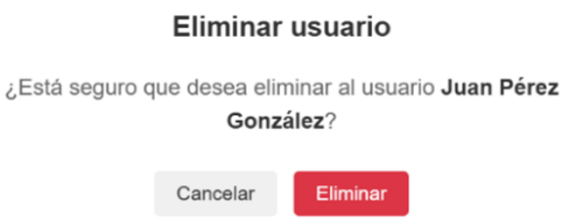
Descripción: Permitirá al administrador eliminar un usuario.
Observaciones: Deberá aparecer una confirmación antes de eliminar.
Interfaz: 

Tabla A.18. Historia de usuario # 20

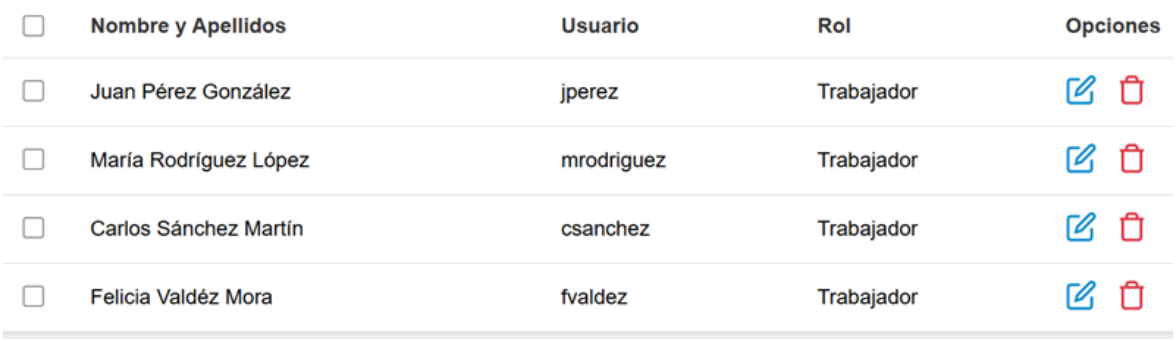
Historia de usuario	
Número: 20	**Nombre:** Listar usuarios
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador visualizar todos los usuarios en forma de tabla con las columnas Nombre y Apellidos, Usuario, Rol y Opciones.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un usuario creado en el sistema.	
Interfaz: 	

Tabla A.19. Historia de usuario # 21

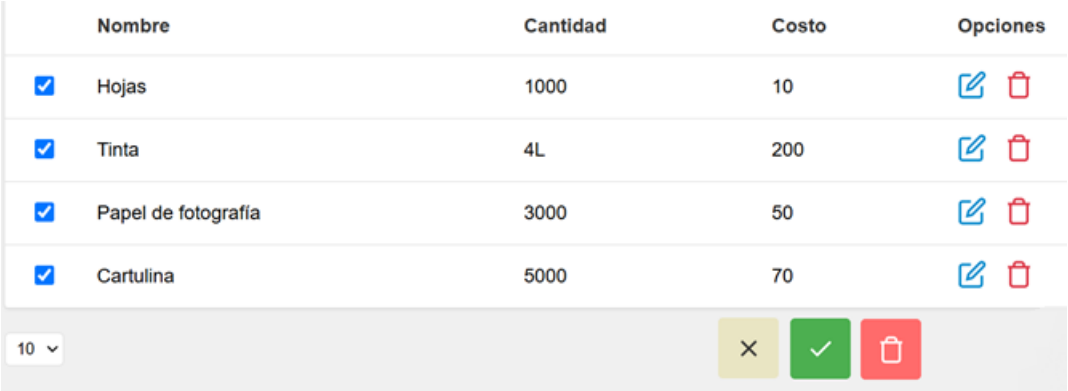
Historia de usuario	
Número: 21	**Nombre:** Selecionar todos los recursos
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador seleccionar todos los recursos.	
Observaciones: El administrador deberá estar autenticado en el sistema.	
Interfaz: 	

Tabla A.20. Historia de usuario # 22

Historia de usuario	
Número: 22	**Nombre:** Deselecionar todos los recursos
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador deseleccionar todos los recursos que estaban seleccionados.	
Observaciones: El administrador deberá estar autenticado en el sistema.	

Continúa en la próxima página

Tabla A.20. Continuación de la página anterior

Interfaz:

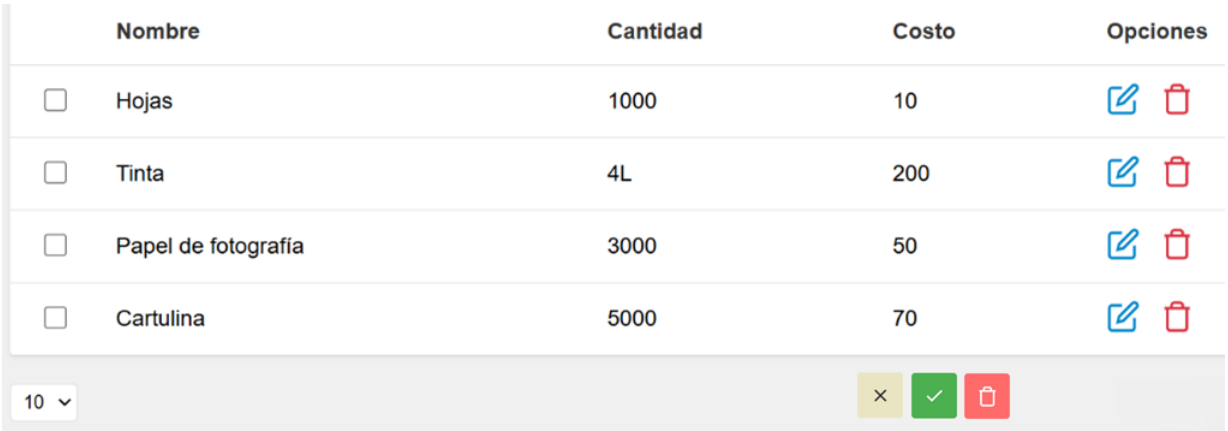


Tabla A.21. Historia de usuario # 23

Historia de usuario	
Número: 23	**Nombre:** Eliminar todos los recursos seleccionados
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador eliminar todos los recursos que estaban seleccionados.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un recurso seleccionado. Deberá aparecer una confirmación antes de eliminar.	
Interfaz: 	

Tabla A.22. Historia de usuario # 24

Historia de usuario	
Número: 24	**Nombre:** Selecionar todos los servicios
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador seleccionar todos los servicios.	
Observaciones: El administrador deberá estar autenticado en el sistema.	
Interfaz: 	

Tabla A.23. Historia de usuario # 25

Historia de usuario	
Número: 25	**Nombre:** Deselecionar todos los servicios
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador deseleccionar todos los servicios que estaban seleccionados.	
Observaciones: El administrador deberá estar autenticado en el sistema.	

Continúa en la próxima página

Tabla A.23. Continuación de la página anterior

Interfaz:

Tabla A.24. Historia de usuario # 26

Historia de usuario	
Número: 26	**Nombre:** Eliminar todos los servicios seleccionados
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador eliminar todos los servicios que estaban seleccionados.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un servicio seleccionado. Deberá aparecer una confirmación antes de eliminar.	

Continúa en la próxima página

Tabla A.24. Continuación de la página anterior

Interfaz:

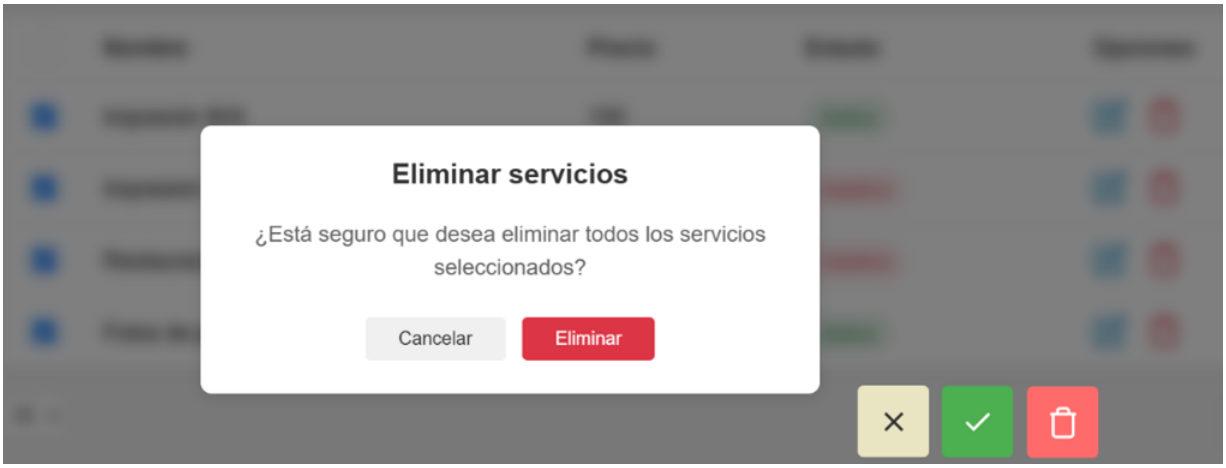


Tabla A.25. Historia de usuario # 27

Historia de usuario	
Número: 27	**Nombre:** Selecionar todos los usuarios
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador eliminar todos los usuarios que estaban seleccionados.	
Observaciones: El administrador deberá estar autenticado en el sistema.	
Interfaz:	

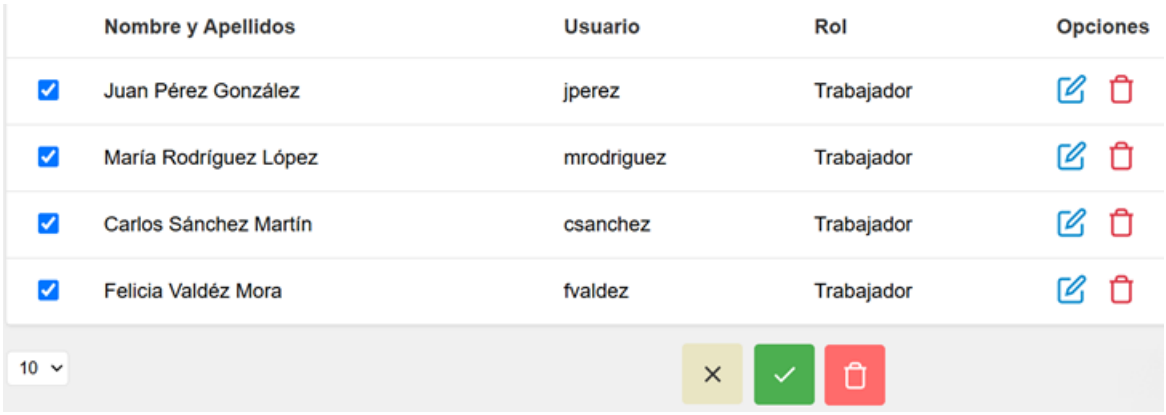


Tabla A.26. Historia de usuario # 28

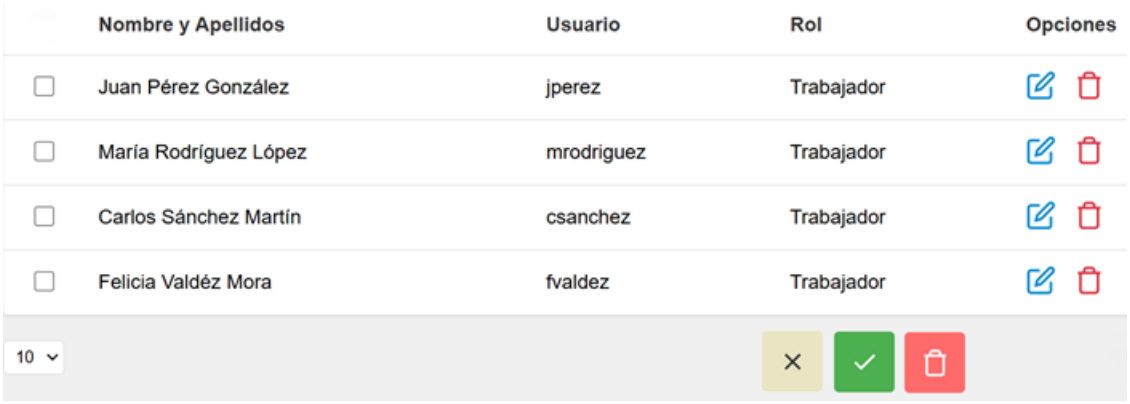
Historia de usuario	
Número: 28	**Nombre:** Deselecionar todos los usuarios
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador deseleccionar todos los usuarios que estaban marcados.	
Observaciones: El administrador deberá estar autenticado en el sistema.	
Interfaz: 	

Tabla A.27. Historia de usuario # 29

Historia de usuario	
Número: 29	**Nombre:** Eliminar todos los usuarios seleccionados
Usuario: Administrador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Stefanni de la Caridad López González	
Descripción: Permitirá al administrador eliminar todos los usuarios que estaban seleccionados.	
Observaciones: El administrador deberá estar autenticado en el sistema. Deberá existir al menos un usurario seleccionado. Deberá aparecer una confirmación antes de eliminar.	

Continúa en la próxima página

Tabla A.27. Continuación de la página anterior

Interfaz:

A.4. Tarjetas CRC

Tabla A.28. Tarjeta CRC # 2

Tarjeta CRC	
Clase: CustomUser	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla A.28. Continuación de la página anterior

• create(): Se utiliza para instanciar y persistir un nuevo objeto en la base de datos. • save(): Guarda el estado actual del objeto en la base de datos, ya sea creando un nuevo registro o actualizando uno existente. • get(): Permite recuperar un único objeto de la base de datos que coincide con los criterios especificados, lanzando una excepción si no se encuentra ninguno. • all(): Devuelve un conjunto de todos los objetos de la clase almacenados en la base de datos. • filter(): Permite consultar la base de datos y devuelve un conjunto de objetos que cumplen con los criterios de filtrado especificados. • delete(): Elimina el objeto de la base de datos, asegurando que ya no esté disponible en futuras consultas.	• WorkerService

Tabla A.29. Tarjeta CRC # 3

Tarjeta CRC	
Clase: Resource	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla A.29. Continuación de la página anterior

• create(): Se utiliza para instanciar y persistir un nuevo objeto en la base de datos. • save(): Guarda el estado actual del objeto en la base de datos, ya sea creando un nuevo registro o actualizando uno existente. • get(): Permite recuperar un único objeto de la base de datos que coincide con los criterios especificados, lanzando una excepción si no se encuentra ninguno. • all(): Devuelve un conjunto de todos los objetos de la clase almacenados en la base de datos. • filter(): Permite consultar la base de datos y devuelve un conjunto de objetos que cumplen con los criterios de filtrado especificados. • delete(): Elimina el objeto de la base de datos, asegurando que ya no esté disponible en futuras consultas.	• ServiceResource

Tabla A.30. Tarjeta CRC # 4

Tarjeta CRC	
Clase: ServiceResource	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla A.30. Continuación de la página anterior

• create(): Se utiliza para instanciar y persistir un nuevo objeto en la base de datos. • save(): Guarda el estado actual del objeto en la base de datos, ya sea creando un nuevo registro o actualizando uno existente. • get(): Permite recuperar un único objeto de la base de datos que coincide con los criterios especificados, lanzando una excepción si no se encuentra ninguno. • all(): Devuelve un conjunto de todos los objetos de la clase almacenados en la base de datos. • filter(): Permite consultar la base de datos y devuelve un conjunto de objetos que cumplen con los criterios de filtrado especificados. • delete(): Elimina el objeto de la base de datos, asegurando que ya no esté disponible en futuras consultas.	• Resource • Service

Tabla A.31. Tarjeta CRC # 5

Tarjeta CRC	
Clase: WorkerService	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla A.31. Continuación de la página anterior

• create(): Se utiliza para instanciar y persistir un nuevo objeto en la base de datos. • save(): Guarda el estado actual del objeto en la base de datos, ya sea creando un nuevo registro o actualizando uno existente. • get(): Permite recuperar un único objeto de la base de datos que coincide con los criterios especificados, lanzando una excepción si no se encuentra ninguno. • all(): Devuelve un conjunto de todos los objetos de la clase almacenados en la base de datos. • filter(): Permite consultar la base de datos y devuelve un conjunto de objetos que cumplen con los criterios de filtrado especificados. • delete(): Elimina el objeto de la base de datos, asegurando que ya no esté disponible en futuras consultas.	• CustomUser • Service

A.5. Tareas de Ingeniería (TI)

A.5.1. Tareas de Ingeniería para la Iteración 1

Tabla A.32. Tarea de desarrollo # 7

Tarea	
Número de tarea: 7	**Número de Historia de usuario:** 3
Nombre de la tarea: Registrar recurso	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 23 de septiembre de 2024	**Fecha de fin:** 25 de septiembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite registrar un nuevo recurso en el sistema.	

Tabla A.33. Tarea de desarrollo # 8

Tarea	
Número de tarea: 8	**Número de Historia de usuario:** 4
Nombre de la tarea: Modificar recurso	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 26 de septiembre de 2024	**Fecha de fin:** 27 de septiembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite modificar la información de un recurso ya existente en el sistema.	

Tabla A.34. Tarea de desarrollo # 9

Tarea	
Número de tarea: 9	**Número de Historia de usuario:** 5
Nombre de la tarea: Registrar servicio	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 28 de septiembre de 2024	**Fecha de fin:** 30 de septiembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite registrar un nuevo servicio a brindar en el taller.	

Tabla A.35. Tarea de desarrollo # 10

Tarea	
Número de tarea: 10	**Número de Historia de usuario:** 6
Nombre de la tarea: Modificar servicio	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 1 de octubre de 2024	**Fecha de fin:** 2 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite modificar la información de un servicio a prestar, ya existente en el sistema.	

Tabla A.36. Tarea de desarrollo # 11

Tarea	
Número de tarea: 11	**Número de Historia de usuario:** 7
Nombre de la tarea: Ofrecer sercvicio	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 3 de octubre de 2024	**Fecha de fin:** 5 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	

Continúa en la próxima página

Tabla A.36. Continuación de la página anterior

Descripción: Se implementa una funcionalidad que permite ofrecer un servicio.

Tabla A.37. Tarea de desarrollo # 12

Tarea	
Número de tarea: 12	**Número de Historia de usuario:** 8
Nombre de la tarea: Visualizar ganancia total en una fecha determinada	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 6 de octubre de 2024	**Fecha de fin:** 8 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite consultar la ganancia total obtenida por la prestación de servicios en una fecha determinada.	

Tabla A.38. Tarea de desarrollo # 13

Tarea	
Número de tarea: 13	**Número de Historia de usuario:** 9
Nombre de la tarea: Visualizar gasto total de recursos en una fecha determinada	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 9 de octubre de 2024	**Fecha de fin:** 11 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite consultar el gasto total de recursos en una fecha determinada.	

Tabla A.39. Tarea de desarrollo # 14

Tarea	
Número de tarea: 14	**Número de Historia de usuario:** 10
Nombre de la tarea: Visualizar servicios ofrecidos en una fecha determinada	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 12 de octubre de 2024	**Fecha de fin:** 14 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite consultar los servicios ofrecidos en una fecha determinada en forma de lista.	

A.5.2. Tareas de Ingeniería para la Iteración 2

Tabla A.40. Tarea de desarrollo # 15

Tarea	
Número de tarea: 15	**Número de Historia de usuario:** 13
Nombre de la tarea: Buscar recursos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 20 de octubre de 2024	**Fecha de fin:** 21 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite realizar búsquedas sobre los recursos.	

Tabla A.41. Tarea de desarrollo # 16

Tarea	
Número de tarea: 16	**Número de Historia de usuario:** 14
Nombre de la tarea: Eliminar servicio.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 22 de octubre de 2024	**Fecha de fin:** 23 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite eliminar un servicio a prestar.	

Tabla A.42. Tarea de desarrollo # 17

Tarea	
Número de tarea: 17	**Número de Historia de usuario:** 15
Nombre de la tarea: Listar servicios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 24 de octubre de 2024	**Fecha de fin:** 25 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite visualizar los servicios a brindar en el taller en forma de lista.	

Tabla A.43. Tarea de desarrollo # 18

Tarea	
Número de tarea: 18	**Número de Historia de usuario:** 16
Nombre de la tarea: Buscar servicios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 26 de octubre de 2024	**Fecha de fin:** 27 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	

Continúa en la próxima página

Tabla A.43. Continuación de la página anterior

Descripción: Se implementa una funcionalidad que permite realizar búsquedas sobre los servicios.

Tabla A.44. Tarea de desarrollo # 19

Tarea	
Número de tarea: 19	**Número de Historia de usuario:** 17
Nombre de la tarea: Cerrar sesión	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 28 de octubre de 2024	**Fecha de fin:** 29 de octubre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite cerrar la sesión en el sistema de forma segura.	

Tabla A.45. Tarea de desarrollo # 20

Tarea	
Número de tarea: 20	**Número de Historia de usuario:** 18
Nombre de la tarea: Modificar usuario.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 30 de octubre de 2024	**Fecha de fin:** 1 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite modificar los datos de un usuario ya existente en el sistema.	

Tabla A.46. Tarea de desarrollo # 21

Tarea	
Número de tarea: 21	**Número de Historia de usuario:** 19
Nombre de la tarea: Eliminar usuario	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 2 de noviembre de 2024	**Fecha de fin:** 3 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite eliminar un usuario del sistema.	

Tabla A.47. Tarea de desarrollo # 22

Tarea

Continúa en la próxima página

Tabla A.47. Continuación de la página anterior

Número de tarea: 22	**Número de Historia de usuario:** 20
Nombre de la tarea: Listar usuarios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 4 de noviembre de 2024	**Fecha de fin:** 5 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite visualizar los servicios del sistema en forma de lista.	

A.5.3. Tareas de Ingeniería para la Iteración 3

Tabla A.48. Tarea de desarrollo # 23

Tarea	
Número de tarea: 23	**Número de Historia de usuario:** 23
Nombre de la tarea: Eliminar todos los recursos seleccionados.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 8 de noviembre de 2024	**Fecha de fin:** 8 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite eliminar a la vez todos los recursos que estaban seleccionados.	

Tabla A.49. Tarea de desarrollo # 24

Tarea	
Número de tarea: 24	**Número de Historia de usuario:** 24
Nombre de la tarea: Seleccionar todos los servicios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 9 de noviembre de 2024	**Fecha de fin:** 9 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite seleccionar a la vez todos los servicios.	

Tabla A.50. Tarea de desarrollo # 25

Tarea	
Número de tarea: 25	**Número de Historia de usuario:** 25
Nombre de la tarea: Deseleccionar todos los servicios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 10 de noviembre de 2024	**Fecha de fin:** 10 de noviembre de 2024

Continúa en la próxima página

Tabla A.50. Continuación de la página anterior

Programador responsable: Stefanni de la Caridad López González
Descripción: Se implementa una funcionalidad que permite deseleccionar a la vez todos los servicios que estaban seleccionados.

Tabla A.51. Tarea de desarrollo # 26

Tarea	
Número de tarea: 26	**Número de Historia de usuario:** 26
Nombre de la tarea: Eliminar todos los servicios seleccionados.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 11 de noviembre de 2024	**Fecha de fin:** 11 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite eliminar a la vez todos los servicios que estaban seleccionados.	

Tabla A.52. Tarea de desarrollo # 27

Tarea	
Número de tarea: 27	**Número de Historia de usuario:** 27
Nombre de la tarea: Seleccionar todos los usuarios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 12 de noviembre de 2024	**Fecha de fin:** 12 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite seleccionar a la vez todos los usuarios.	

Tabla A.53. Tarea de desarrollo # 28

Tarea	
Número de tarea: 28	**Número de Historia de usuario:** 22
Nombre de la tarea: Deseleccionar todos los usuarios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 13 de noviembre de 2024	**Fecha de fin:** 13 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite deseleccionar a la vez todos los usuarios que estaban seleccionados.	

Tabla A.54. Tarea de desarrollo # 29

Tarea	
Número de tarea: 29	**Número de Historia de usuario:** 29
Nombre de la tarea: Eliminar todos los usuarios seleccionados.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 14 de noviembre de 2024	**Fecha de fin:** 14 de noviembre de 2024
Programador responsable: Stefanni de la Caridad López González	
Descripción: Se implementa una funcionalidad que permite eliminar a la vez todos los usuarios que estaban seleccionados.	

A.6. Pruebas unitarias

A.6.1. Pruebas unitarias para la Iteración 1

```
56    def test_register_successful(self):
57        print('-'*100,'\n','testing register successful initiated')
58        data = {
59            'username': 'newuser',
60            'password': 'Test@2024',
61            'first_name': 'New',
62            'last_name': 'User',
63            'role': 'worker'
64        }
65        response = self.client.post(self.register_url, data)
66        self.assertEqual(response.status_code, status.HTTP_200_OK)
67        self.assertIn('access', response.data)
68        self.assertTrue(get_user_model().objects.filter(username='newuser').exists())
69        print('register successful test passed')
70
```

Figura A.1. Caso de prueba unitaria Registrar usuario.

```python
def test_update_resource_successful(self):
    print('-'*100, '\n', 'testing update resource successful initiated')
    resource = Resource.objects.create(
        name='Original Resource',
        quantity=5,
        price=9.99
    )
    data = {
        'id': resource.id,
        'name': 'Updated Resource',
        'quantity': 15,
        'price': '19.99'
    }
    response = self.client.post('/api/update_resource/', data, format='json')
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    print('update resource successful test passed')
```

Figura A.2. Caso de prueba unitaria Modificar recurso.

```python
def test_create_service(self):
    print('-'*100, '\n', 'testing create service initiated')
    data = {
        'name': 'Test Service',
        'cost': '29.99',
        'description': 'Test Description',
        'resources': [
            {'resource_id': self.resource1.id},
            {'resource_id': self.resource2.id}
        ]
    }
    response = self.client.post('/api/create_service/', data, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    print('create service test passed')
```

Figura A.3. Caso de prueba unitaria Registrar servicio.

```python
    def test_update_service(self):
        print('-'*100, '\n', 'testing update service initiated')
        service = Service.objects.create(
            name='Original Service',
            descripcion='Original Description',
            cost='19.99'
        )
        data = {
            'id': service.id,
            'name': 'Updated Service',
            'cost': '39.99',
            'description': 'Updated Description',
            'resources': [
                {'resource_id': self.resource1.id}
            ]
        }
        response = self.client.post('/api/update_service/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        print('update service test passed')

```

Figura A.4. Caso de prueba unitaria Modificar servicio.

A.6.2. Pruebas unitarias para la Iteración 2

```python

    def test_get_resources_successful(self):
        print('-'*100,'\n','testing get resources successful initiated')
        response = self.client.post(self.url, {'search': '', 'page': 1, 'per_page': 10}, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data['resources']), 1)
        print('get resources successful test passed')
```

Figura A.5. Caso de prueba unitaria Listar recursos.

```python
    def test_get_resources_with_search(self):
        print('-'*100, '\n', 'testing get resources with search initiated')

        Resource.objects.create(name='Special Resource', quantity=5, price=9.99)
        Resource.objects.create(name='Normal Resource', quantity=5, price=9.99)

        data = {
            'search': 'Special',
            'page': 1,
            'per_page': 10
        }
        response = self.client.post('/api/get_resources/', data, format='json')

        self.assertEqual(len(response.data['resources']), 2)
        print('get resources with search test failed as expected')
```

Figura A.6. Caso de prueba unitaria Buscar recursos.

```python
    def test_get_all_services(self):
        print('-'*100, '\n', 'testing get all services initiated')
        Service.objects.create(
            name='Service 1',
            descripcion='Description 1',
            cost='19.99'
        )
        Service.objects.create(
            name='Service 2',
            descripcion='Description 2',
            cost='29.99'
        )
        response = self.client.get('/api/get_all_services/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data['services']), 2)
        print('get all services test passed')

```

Figura A.7. Caso de prueba unitaria Listar servicios.

```python
    def test_get_services_with_search(self):
        print('-'*100, '\n', 'testing get services with search initiated')
        Service.objects.create(
            name='Special Service',
            descripcion='Special Description',
            cost='19.99'
        )
        Service.objects.create(
            name='Normal Service',
            descripcion='Normal Description',
            cost='29.99'
        )
        response = self.client.get('/api/get_services/', {'search': 'Special'})
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data['services']), 1)
        print('get services with search test passed')

```

Figura A.8. Caso de prueba unitaria Buscar servicios.

```python
    def test_update_user_successful(self):
        print('-'*100,'\n','testing update user successful initiated')
        data = {
            'id': self.user.id,
            'username': 'updated_admin',
            'password': 'NewPass@2024',
            'first_name': 'Updated',
            'last_name': 'Admin',
            'role': 'admin'
        }
        response = self.client.put(self.url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        updated_user = get_user_model().objects.get(id=self.user.id)
        self.assertEqual(updated_user.username, 'updated_admin')
        print('update user successful test passed')
```

Figura A.9. Caso de prueba Modificar usuario.

```python
    def test_delete_user_successful(self):
        print('-'*100,'\n','testing delete user successful initiated')
        test_user = get_user_model().objects.create_user(
            username='test_delete',
            password='Test@2024',
            role='worker'
        )
        data = {'id': test_user.id}
        response = self.client.post(self.url, data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertFalse(get_user_model().objects.filter(id=test_user.id).exists())
        print('delete user successful test passed')
```

Figura A.10. Caso de prueba Eliminar usuario.

```python
def test_get_users_successful(self):
    print('-'*100, '\n', 'testing get users successful initiated')
    self.client.force_authenticate(user=self.admin)
    response = self.client.get('/api/get_users/', format='json')
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertTrue('users' in response.data)
    print('get users successful test passed')
```

Figura A.11. Caso de prueba Listar usuarios.

A.6.3. Pruebas unitarias para la Iteración 3

```python
def test_deselect_all_resources(self):
    print('-'*100, '\n', 'testing deselect all resources initiated')

    response = self.client.post('/api/resources/deselect/', {
        'selected': []
    }, format='json')

    self.assertEqual(response.status_code, status.HTTP_200_OK)
    print('deselect all resources test passed')
```

Figura A.12. Caso de prueba unitaria Deseleccionar todos los recursos.

```python
def test_select_all_services(self):
    print('-'*100, '\n', 'testing select all services initiated')

    Service.objects.create(
        name='Service 1',
        descripcion='Description 1',
        cost='19.99'
    )
    Service.objects.create(
        name='Service 2',
        descripcion='Description 2',
        cost='29.99'
    )

    response = self.client.get('/api/get_all_services/')

    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(len(response.data['services']), 2)
    print('select all services test passed')

```

Figura A.13. Caso de prueba unitaria Seleccionar todos los servicios.

```python
def test_deselect_all_services(self):
    print('-'*100, '\n', 'testing deselect all services initiated')

    response = self.client.post('/api/services/deselect/', {
        'selected': []
    }, format='json')

    self.assertEqual(response.status_code, status.HTTP_200_OK)
    print('deselect all services test passed')
```

Figura A.14. Caso de prueba unitaria Deseleccionar todos los servicios.

```python
def test_delete_selected_services(self):
    print('-'*100, '\n', 'testing delete selected services initiated')

    service1 = Service.objects.create(
        name='Service to Delete 1',
        descripcion='To be deleted 1',
        cost='19.99'
    )
    service2 = Service.objects.create(
        name='Service to Delete 2',
        descripcion='To be deleted 2',
        cost='29.99'
    )

    response = self.client.post('/api/services/delete-many/', {
        'ids': [service1.id, service2.id]
    }, format='json')

    self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
    self.assertEqual(Service.objects.filter(id__in=[service1.id, service2.id]).count(), 0)
    print('delete selected services test passed')
```

Figura A.15. Caso de prueba unitaria Eliminar todos los servicios seleccionados.

```python
def test_deselect_all_users(self):
    print('-'*100, '\n', 'testing deselect all users initiated')

    response = self.client.post('/api/users/deselect/', {
        'selected': []
    }, format='json')

    self.assertEqual(response.status_code, status.HTTP_200_OK)
    print('deselect all users test passed')
```

Figura A.16. Caso de prueba unitaria Deseleccionar todos los usuarios.

```python
def test_delete_selected_users(self):
    print('-'*100, '\n', 'testing delete selected users initiated')

    data1 = {
        'username': 'deleteuser1',
        'password': 'Test2024',
        'first_name': 'Delete',
        'last_name': 'User1',
        'role': 'worker'
    }
    data2 = {
        'username': 'deleteuser2',
        'password': 'Test2024',
        'first_name': 'Delete',
        'last_name': 'User2',
        'role': 'worker'
    }

    user1_response = self.client.post(self.register_url, data1)
    user2_response = self.client.post(self.register_url, data2)

    user1_id = user1_response.data['user']['id']
    user2_id = user2_response.data['user']['id']

    response = self.client.post('/api/users/delete-many/', {
        'ids': [user1_id, user2_id]
    }, format='json')

    self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
    self.assertEqual(get_user_model().objects.filter(id__in=[user1_id, user2_id]).count(), 0)
    print('delete selected users test passed')
```

Figura A.17. Caso de prueba unitaria Eliminar todos los usuarios seleccionados.

A.7. Pruebas de aceptación

A.7.1. Pruebas de aceptación para la Iteración 1

Tabla A.55. Prueba de aceptación # 7

Caso de prueba de aceptación	
Código: HU3_CPA3	**Historia de usuario:** 3
Nombre: Registrar recurso	
Descripción: Prueba para la funcionalidad Registrar recurso.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema.	

Continúa en la próxima página

Tabla A.55. Continuación de la página anterior

Pasos de ejecución: • El administrador debe estar autenticado en el sistema. • El administrador selecciona el botón con ícono de adicionar. • Se despliega un formulario con los campos requeridos (nombre, cantidad, costo). • El administrador llena todos los campos obligatorios con información válida. • El administrador presiona el botón Guardar.
Resultados esperados: • Si todos los datos son válidos, el sistema registra el nuevo recurso y lo incluye en la lista de recursos visibles • Si algún campo obligatorio no se llena o contiene datos inválidos, el sistema muestra un mensaje de error indicando el problema.

Tabla A.56. Prueba de aceptación # 8

Caso de prueba de aceptación	
Código: HU4_CPA4	**Historia de usuario:** 4
Nombre: Modificar recurso	
Descripción: Prueba para la funcionalidad Modificar recurso.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un recurso registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Recursos. • El administrador localiza al recurso que desea modificar en la lista de recursos. • El administrador selecciona el ícono de Editar asociado al recurso. • El sistema despliega un formulario con los datos actuales del recurso. • El administrador modifica los campos requeridos (nombre, cantidad, costo) con información válida. • El administrador presiona el botón Guardar.	
Resultados esperados: • Si los datos modificados son válidos, el sistema actualiza la información del recurso y refleja los cambios en la lista de recursos. • Si algún campo obligatorio no se llena o contiene datos inválidos, el sistema muestra un mensaje de error indicando el problema.	

Tabla A.57. Prueba de aceptación # 9

Caso de prueba de aceptación	
Código: HU5_CPA5	**Historia de usuario:** 5
Nombre: Registrar servicio	
Descripción: Prueba para la funcionalidad Registrar servicio.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema.	
Pasos de ejecución: • El administrador debe estar autenticado en el sistema. • El administrador selecciona el botón con ícono de adicionar. • Se despliega un formulario con los campos requeridos (nombre, descripción, precio, recursos que consume y estado). • El administrador llena todos los campos obligatorios con información válida. • El administrador presiona el botón Guardar.	
Resultados esperados: • Si todos los datos son válidos, el sistema registra el nuevo servicio y lo incluye en la lista de servicios visibles. • Si algún campo obligatorio no se llena o contiene datos inválidos, el sistema muestra un mensaje de error indicando el problema.	

Tabla A.58. Prueba de aceptación # 10

Caso de prueba de aceptación	
Código: HU6_CPA6	**Historia de usuario:** 6
Nombre: Modificar servicio	
Descripción: Prueba para la funcionalidad Modificar servicio.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un recurso registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Servicios. • El administrador localiza al recurso que desea modificar en la lista de servicios. • El administrador selecciona el ícono de Editar asociado al servicio. • El sistema despliega un formulario con los datos actuales del servicio. • El administrador modifica los campos requeridos (nombre, descripción, precio, recursos que consume y estado). • El administrador presiona el botón Guardar.	

Continúa en la próxima página

Tabla A.58. Continuación de la página anterior

Resultados esperados: • Si los datos modificados son válidos, el sistema actualiza la información del servicio y refleja los cambios en la lista de servicios. • Si algún campo obligatorio no se llena o contiene datos inválidos, el sistema muestra un mensaje de error indicando el problema.

Tabla A.59. Prueba de aceptación # 11

Caso de prueba de aceptación	
Código: HU7_CPA7	**Historia de usuario:** 7
Nombre: Ofrecer servicio	
Descripción: Prueba para la funcionalidad Ofrecer servicio.	
Condiciones de ejecución: • El trabajador debe estar autenticado en el sistema. • El servicio debe estar habilitado por el administrador.	
Pasos de ejecución: • El trabajador accede a la sección Servicios disponibles. • Selecciona el servicio a ofrecer. • Introduce la cantidad requerida en el campo correspondiente. • Presiona el botón Registrar servicio.	
Resultados esperados: • El sistema registra la prestación del servicio y actualiza la información correspondiente. • Si algún dato no es válido, se muestra un mensaje de error.	

Tabla A.60. Prueba de aceptación # 12

Caso de prueba de aceptación	
Código: HU8_CPA8	**Historia de usuario:** 8
Nombre: Visualizar la ganancia total en una fecha determinada.	
Descripción: Prueba para la funcionalidad Visualizar ganancia total en una fecha determinada.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir registros de servicios prestados.	

Continúa en la próxima página

Tabla A.60. Continuación de la página anterior

Pasos de ejecución: • El administrador accede a la sección Reporte de ganancias. • Selecciona una fecha en el calendario. • Presiona el botón Filtrar.
Resultados esperados: • El sistema muestra las ganancias totales del día seleccionado, con información detallada por servicio y trabajador que lo realizó. • Si no hay registros para esa fecha, se muestra un mensaje No hay ganancias registradas para esta fecha.

Tabla A.61. Prueba de aceptación # 13

Caso de prueba de aceptación	
Código: HU9_CPA9	**Historia de usuario:** 9
Nombre: Visualizar gasto total de recursos en una fecha determinada.	
Descripción: Prueba para la funcionalidad Visualizar gasto total de recursos en una fecha determinada.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir registros de servicios ofrecidos que gasten recursos.	
Pasos de ejecución: • El administrador accede a la sección Reporte de gastos. • Selecciona una fecha en el calendario. • Presiona el botón Filtrar.	
Resultados esperados: • El sistema muestra el gasto total en recursos del día seleccionado, con información detallada por tipo de recurso, cantidad usada y costo. • Si no hay registros para esa fecha, se muestra un mensaje No hay gastos registradas para esta fecha.	

Tabla A.62. Prueba de aceptación # 14

Caso de prueba de aceptación	
Código: HU10_CPA10	**Historia de usuario:** 10
Nombre: Visualizar los servicios ofrecidos en una fecha determinada.	

Continúa en la próxima página

Tabla A.62. Continuación de la página anterior

Descripción: Prueba para la funcionalidad Visualizar servicios ofrecidos en una fecha determinada.
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir registros de servicios ofrecidos.
Pasos de ejecución: • El administrador accede a la sección Servicios ofrecidos. • Selecciona una fecha en el calendario. • Presiona el botón Filtrar.
Resultados esperados: • El sistema muestra los servicios ofrecidos con su información correspondiente. • Si no existen registros, se muestra un mensaje No hay servicios ofrecidos.

A.7.2. Pruebas de aceptación para la Iteración 2

Tabla A.63. Prueba de aceptación # 15

Caso de prueba de aceptación	
Código: HU13_CPA13	**Historia de usuario:** 13
Nombre: Buscar recursos.	
Descripción: Prueba para la funcionalidad Buscar recursos.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un recurso registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Recursos. • Introduce un criterio de búsqueda (nombre del recurso) en el campo correspondiente. • Presiona el botón de Buscar.	
Resultados esperados: • Si existen coincidencias, se muestra una lista con los recursos que cumplen el criterio. • Si no hay recursos registrados, se muestra un mensaje diciendo que no hay criterios coincidentes.	

Tabla A.64. Prueba de aceptación # 16

Caso de prueba de aceptación	
Código: HU14_CPA14	**Historia de usuario:** 14
Nombre: Eliminar servicio.	
Descripción: Prueba para la funcionalidad Eliminar servicio.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un servicio registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Servicios. • El administrador localiza el servicio que desea eliminar en la lista de servicios. • El administrador selecciona el ícono de papelera asociada al servicio. • El sistema despliega una ventana emergente solicitando la confirmación de eliminación con el mensaje: ¿Está seguro que desea eliminar este servicio? • El administrador confirma la eliminación presionando el botón Eliminar.	
Resultados esperados: • Si el administrador confirma, el sistema elimina el servicio seleccionado y lo remueve de la lista de servicios. • Si el administrador cancela la acción, el sistema mantiene intacta la información del servicio.	

Tabla A.65. Prueba de aceptación # 17

Caso de prueba de aceptación	
Código: HU15_CPA15	**Historia de usuario:** 15
Nombre: Listar servicios.	
Descripción: Prueba para la funcionalidad Listar servicios.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un servicio registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Servicios. • El sistema muestra una tabla con las columnas Nombre, Precio, Estado y Opciones.	
Resultados esperados: • La lista muestra todos los recursos registrados con la información correspondiente. • Si no hay servicios registrados, se muestra un mensaje No hay recursos disponibles.	

Tabla A.66. Prueba de aceptación # 18

Caso de prueba de aceptación	
Código: HU16_CPA16	**Historia de usuario:** 16
Nombre: Buscar servicios.	
Descripción: Prueba para la funcionalidad Buscar servicios.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un servicio registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Servicios. • Introduce un criterio de búsqueda (nombre del servicio) en el campo correspondiente. • Presiona el botón de Buscar.	
Resultados esperados: • Si existen coincidencias, se muestra una lista con los servicios que cumplen el criterio. • Si no hay servicios registrados, se muestra un mensaje diciendo que no hay criterios coincidentes.	

Tabla A.67. Prueba de aceptación # 19

Caso de prueba de aceptación	
Código: HU17_CPA17	**Historia de usuario:** 17
Nombre: Cerrar sesión.	
Descripción: Prueba para la funcionalidad Cerrar sesión.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema.	
Pasos de ejecución: • El administrador accede al botón de Cerrar sesión. • Presiona el botón de Cerrar sesión.	
Resultados esperados: • Si presiona Aceptar, se cierra la sesión. • Si presiona Cancelar, se cancela la acción.	

Tabla A.68. Prueba de aceptación # 20

Caso de prueba de aceptación	
Código: HU18_CPA18	**Historia de usuario:** 18

Continúa en la próxima página

Tabla A.68. Continuación de la página anterior

Nombre: Modificar usuario
Descripción: Prueba para la funcionalidad Modificar usuario.
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un usuario registrado en el sistema.
Pasos de ejecución: • El administrador accede a la sección Usuarios. • El administrador localiza al usuario que desea modificar en la lista de usuarios. • El administrador selecciona el ícono de Editar asociado al usuario. • El sistema despliega un formulario con los datos actuales del usuario. • El administrador modifica los campos requeridos (nombre, apellidos, nombre de usuario, contraseña) con información válida. • El administrador presiona el botón Guardar.
Resultados esperados: • Si los datos modificados son válidos, el sistema actualiza la información del usuario y refleja los cambios en la lista de usuarios. • Si algún campo obligatorio no se llena o contiene datos inválidos, el sistema muestra un mensaje de error indicando el problema.

Tabla A.69. Prueba de aceptación # 21

Caso de prueba de aceptación	
Código: HU19_CPA19	**Historia de usuario:** 19
Nombre: Eliminar usuario.	
Descripción: Prueba para la funcionalidad Eliminar usuario.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un usuario registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Usuarios. • El administrador localiza el usuario que desea eliminar en la lista de usuarios. • El administrador selecciona el ícono de papelera asociado al usuario. • El sistema despliega una ventana emergente solicitando la confirmación de eliminación con el mensaje: ¿Está seguro que desea eliminar este usuario? • El administrador confirma la eliminación presionando el botón Eliminar.	

Continúa en la próxima página

Tabla A.69. Continuación de la página anterior

Resultados esperados: • Si el administrador confirma, el sistema elimina el usuario seleccionado y lo remueve de la lista de usuarios. • Si el administrador cancela la acción, el sistema mantiene intacta la información del usuario.

Tabla A.70. Prueba de aceptación # 22

Caso de prueba de aceptación	
Código: HU20_CPA20	**Historia de usuario:** 20
Nombre: Listar usuarios.	
Descripción: Prueba para la funcionalidad Listar usuarios.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Debe existir al menos un usuario registrado en el sistema.	
Pasos de ejecución: • El administrador accede a la sección Usuarios. • El sistema muestra una tabla con las columnas Nombre y Apellidos, Usuario, Rol y Opciones.	
Resultados esperados: • La lista muestra todos los usuarios registrados con la información correspondiente. • Si no hay usuarios registrados, se muestra un mensaje No hay usuarios disponibles.	

A.7.3. Pruebas de aceptación para la Iteración 3

Tabla A.71. Prueba de aceptación # 23

Caso de prueba de aceptación	
Código: HU23_CPA23	**Historia de usuario:** 23
Nombre: Eliminar todos los recursos seleccionados.	
Descripción: Prueba para la funcionalidad Eliminar todos los recursos seleccionados.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir recursos seleccionados.	

Continúa en la próxima página

Tabla A.71. Continuación de la página anterior

Pasos de ejecución: • El administrador accede a la sección Recursos. • El administrador selecciona varios recursos mediante el *checkbox* o con el botón de Seleccionar todos. • El administrador presiona el botón rojo flotante con el ícono de papelera. • El sistema despliega una ventana emergente con el mensaje: ¿Está seguro que desea eliminar todos los recursos seleccionados?. • El administrador confirma la acción presionando el botón Eliminar.
Resultados esperados: • Si el administrador confirma, el sistema elimina a todos los recursos seleccionados. • Si no hay recursos seleccionados, el botón está deshabilitado.

Tabla A.72. Prueba de aceptación # 24

Caso de prueba de aceptación	
Código: HU24_CPA24	**Historia de usuario:** 24
Nombre: Seleccionar todos los servicios.	
Descripción: Prueba para la funcionalidad Seleccionar todos los servicios.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir servicios listados.	
Pasos de ejecución: • El administrador accede a la sección Servicios. • El administrador presiona el botón verde flotante con el ícono de *check*.	
Resultados esperados: • Todos los servicios de la lista quedan seleccionados. • Si no hay servicios listados, el botón está deshabilitado.	

Tabla A.73. Prueba de aceptación # 25

Caso de prueba de aceptación	
Código: HU25_CPA25	**Historia de usuario:** 25
Nombre: Deseleccionar todos los servicios.	
Descripción: Prueba para la funcionalidad Deseleccionar todos los servicios.	

Continúa en la próxima página

Tabla A.73. Continuación de la página anterior

Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir servicios seleccionados.
Pasos de ejecución: • El administrador accede a la sección Servicios. • El administrador presiona el botón gris flotante con el ícono de X.
Resultados esperados: • Todos los servicios seleccionados quedan desmarcados. • Si no hay servicios seleccionados, el botón está deshabilitado.

Tabla A.74. Prueba de aceptación # 26

Caso de prueba de aceptación	
Código: HU26_CPA26	**Historia de usuario:** 26
Nombre: Eliminar todos los servicios seleccionados.	
Descripción: Prueba para la funcionalidad Eliminar todos los servicios seleccionados.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir servicios seleccionados.	
Pasos de ejecución: • El administrador accede a la sección Servicios. • El administrador selecciona varios servicios mediante el *checkbox* o con el botón de Seleccionar todos. • El administrador presiona el botón rojo flotante con el ícono de papelera. • El sistema despliega una ventana emergente con el mensaje: ¿Está seguro que desea eliminar todos los servicios seleccionados?. • El administrador confirma la acción presionando el botón Eliminar.	
Resultados esperados: • Si el administrador confirma, el sistema elimina a todos los servicios seleccionados. • Si no hay servicios seleccionados, el botón está deshabilitado.	

Tabla A.75. Prueba de aceptación # 27

Caso de prueba de aceptación	
Código: HU27_CPA27	**Historia de usuario:** 27

Continúa en la próxima página

Tabla A.75. Continuación de la página anterior

Nombre: Seleccionar todos los usuarios.
Descripción: Prueba para la funcionalidad Seleccionar todos los usuarios.
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir usuarios listados.
Pasos de ejecución: • El administrador accede a la sección Usuarios. • El administrador presiona el botón verde flotante con el ícono de *check*.
Resultados esperados: • Todos los usuarios de la lista quedan seleccionados. • Si no hay usuarios listados, el botón está deshabilitado.

Tabla A.76. Prueba de aceptación # 28

Caso de prueba de aceptación	
Código: HU28_CPA28	**Historia de usuario:** 28
Nombre: Deseleccionar todos los usuarios.	
Descripción: Prueba para la funcionalidad Deseleccionar todos los usuarios.	
Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir usuarios seleccionados.	
Pasos de ejecución: • El administrador accede a la sección Usuarios. • El administrador presiona el botón gris flotante con el ícono de X.	
Resultados esperados: • Todos los usuarios seleccionados quedan desmarcados. • Si no hay usuarios seleccionados, el botón está deshabilitado.	

Tabla A.77. Prueba de aceptación # 29

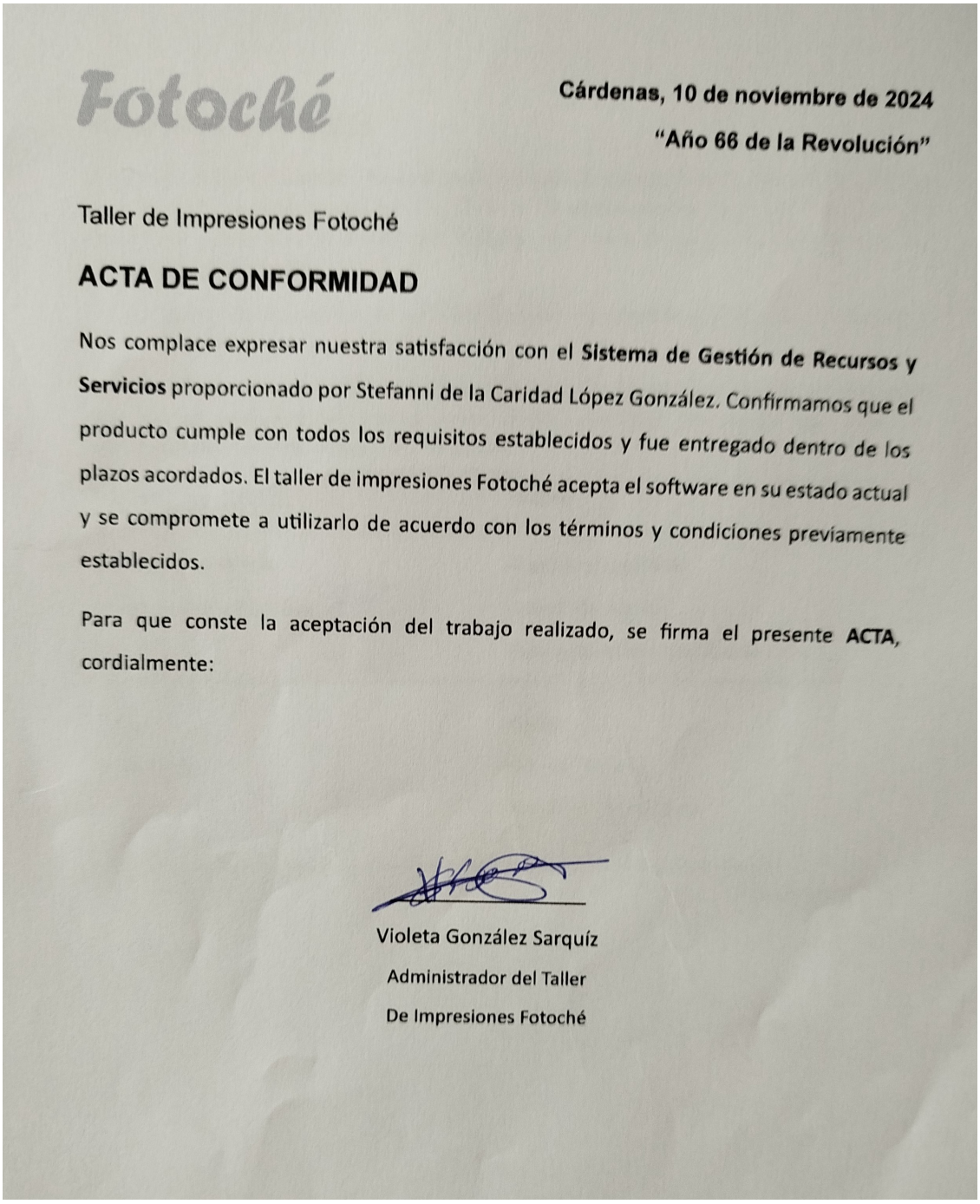
Caso de prueba de aceptación	
Código: HU29_CPA29	**Historia de usuario:** 29
Nombre: Eliminar todos los usuarios seleccionados.	
Descripción: Prueba para la funcionalidad Eliminar todos los usuarios seleccionados.	

Continúa en la próxima página

Tabla A.77. Continuación de la página anterior

Condiciones de ejecución: • El administrador debe estar autenticado en el sistema. • Deben existir usuarios seleccionados.
Pasos de ejecución: • El administrador accede a la sección Usuarios. • El administrador selecciona varios usuarios mediante el *checkbox* o con el botón de Seleccionar todos. • El administrador presiona el botón rojo flotante con el ícono de papelera. • El sistema despliega una ventana emergente con el mensaje: ¿Está seguro que desea eliminar todos los usuarios seleccionados?. • El administrador confirma la acción presionando el botón Eliminar.
Resultados esperados: • Si el administrador confirma, el sistema elimina a todos los usuarios seleccionados. • Si no hay usuarios seleccionados, el botón está deshabilitado.

A.8. Aval del cliente



Fotoché

Cárdenas, 10 de noviembre de 2024

"Año 66 de la Revolución"

Taller de Impresiones Fotoché

ACTA DE CONFORMIDAD

Nos complace expresar nuestra satisfacción con el **Sistema de Gestión de Recursos y Servicios** proporcionado por Stefanni de la Caridad López González. Confirmamos que el producto cumple con todos los requisitos establecidos y fue entregado dentro de los plazos acordados. El taller de impresiones Fotoché acepta el software en su estado actual y se compromete a utilizarlo de acuerdo con los términos y condiciones previamente establecidos.

Para que conste la aceptación del trabajo realizado, se firma el presente **ACTA**, cordialmente:

Violeta González Sarquíz

Administrador del Taller

De Impresiones Fotoché

Figura A.18. Carta de aceptación del cliente.