PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE: SISTEMAS DE INFORMACIÓN

Trabajo de Integración Curricular

Tema:

DESARROLLO DE UN SISTEMA DE INFORMACIÓN PROTOTIPO PARA EL REGISTRO Y CONTROL DEL DESPACHO DE GASOLINA DE PESCA ARTESANAL, CASO DE ESTUDIO “DISTRIBUIDORA DEL PUERTO”.

AUTOR:

Adrián Marcelo Galeas Figueroa

QUITO DM, JUNIO DE 2024

DEDICATORIA

Dedico este trabajo de titulación a tres pilares fundamentales en mi vida: a Dios, a mi madre y a mi primo Amigdio.

A Dios, le extiendo mi más sincera gratitud por haberme dotado de la fortaleza y sabiduría necesarias para enfrentar y superar los retos que se han presentado en mi camino, guiándome con su luz hasta alcanzar este significativo hito en mi vida.

A mi madre, a quien le debo toda mi gratitud por su amor incondicional y constante apoyo. Su presencia ha sido un faro de guía y seguridad en mi vida, facilitando cada paso hacia la realización de mis sueños. Su sacrificio y dedicación son la base de mis logros.

A mi primo Amigdio, extiendo mi profundo agradecimiento por su soporte incansable y su presencia en los momentos más desafiantes de mi carrera. Su apoyo fue esencial, brindándome la fuerza necesaria para superar obstáculos y seguir adelante.

Cada uno de ustedes ha sido fundamental en este viaje, y este logro es también suyo.

AGRADECIMIENTOS

Quisiera expresar mi profundo agradecimiento a todas las personas que han sido parte de mi travesía a lo largo de mi carrera universitaria.

A mi familia y amigos, cuyo apoyo incondicional ha sido un pilar fundamental en la persecución y realización de mis sueños académicos y profesionales.

Extiendo un agradecimiento especial a mi directora de integración curricular, Beatriz Campos, cuyas enseñanzas y consejos han sido esenciales en mi desarrollo académico y personal. Su orientación experta ha jugado un papel decisivo en mi trayectoria educativa.

Agradezco al profesor Jorge Alarcón, cuyas clases fueron cruciales para que descubriera mi pasión por la programación. Su habilidad para transmitir el conocimiento de manera clara y motivadora ha sido fundamental en mi formación académica.

A todos mis profesores, les agradezco por su guía constante y por alentarme a explorar nuevos horizontes en el ámbito tecnológico, enriqueciendo así mi formación y visión profesional.

Finalmente, mi gratitud a la Pontificia Universidad Católica del Ecuador, por brindarme una educación de alta calidad y por ser el escenario donde he podido crecer tanto personal como profesionalmente.

RESUMEN

El presente trabajo de titulación desarrolla un sistema de información para optimizar el registro y control del despacho de gasolina de pesca artesanal en la "Distribuidora del Puerto". El objetivo principal es mejorar la eficiencia operativa y el cumplimiento de las regulaciones mediante la digitalización de procesos manuales. La tesis comienza justificando la necesidad del sistema debido a las deficiencias en los procesos manuales y las estrictas regulaciones gubernamentales. Se establecen objetivos específicos para desarrollar un prototipo funcional que optimice tanto la gestión documental como la operativa. El marco teórico aborda los sistemas de información y las metodologías de desarrollo de software, enfocándose en la Programación Extrema (XP) por su adaptabilidad y colaboración. Se destacan herramientas clave como PlantUML, NextJS y PostgreSQL, fundamentales en el desarrollo del sistema. Los requerimientos del sistema se describen a través de historias de usuario, cubriendo aspectos desde la gestión de sesiones y usuarios hasta la administración de ventas y reportes. Se explica el proceso de desarrollo estructurado en iteraciones, cada una con objetivos claros y pruebas de aceptación, utilizando Microsoft Planner para el seguimiento de tareas. Además, se incluyen diagramas de caso de uso que ilustran la interacción entre componentes del sistema y el diseño de la base de datos, desarrollado en etapas conceptual, lógica y física, para garantizar una gestión eficiente de la información. El documento finaliza subrayando cómo el sistema mejora la eficiencia y el cumplimiento normativo en la distribución de gasolina para pesca artesanal, con recomendaciones para la expansión futura del sistema y mejoras continuas.

Tabla de contenido

Tabla de figuras

CAPÍTULO I: DEFINICIÓN DEL TRABAJO

1.1. TEMA

Desarrollo de un sistema de información prototipo para el registro y control del despacho de gasolina de pesca artesanal, caso de estudio “Distribuidora del Puerto”.

1.2. JUSTIFICACIÓN

La distribución de gasolina de pesca artesanal en el territorio ecuatoriano es un proceso delicado debido a que es un derivado del petróleo subsidiado por el gobierno ecuatoriano, por ello, existe una serie de instituciones públicas encargadas principalmente de controlar la comercialización de esta gasolina con la capacidad de actuar de acuerdo con la ley, sin embargo, no todos los organismos gubernamentales se encuentran presentes en el proceso de venta de gasolina a los pescadores artesanales autorizados.

En el proceso de venta de gasolina para la pesca artesanal, las gasolineras autorizadas, tanto del sector privado como público, abastecen de gasolina a dichos pescadores para sus actividades productivas, no obstante, dichos establecimientos deben acatar de manera estricta las regulaciones y normativas establecidas por los entes de control correspondientes, para no incurrir en faltas a la ley orgánica y a la imposición de multas por irregularidades durante el proceso en cuestión.

Por lo supuesto, se pretende apoyar a los distribuidores de gasolina de pesca artesanal autorizados a lo largo de la costa ecuatoriana a implementar herramientas tecnológicas diseñadas para optimizar la gestión de los documentos provistos y solicitados por las instituciones de control durante el despacho de la gasolina al cliente final.

1.3. PLANTEAMIENTO DEL PROBLEMA

La Agencia de Regulación y Control de Energía y Recursos Naturales no Renovables (ARCERNNR)[1] es uno de los principales organismos gubernamentales encargados de controlar la distribución de gasolina para pesca artesanal en el Ecuador que avala el cumplimiento de Acuerdo Interinstitucional "Instructivo para la Distribución de

[1] Encargada de regular, controlar, fiscalizar, y auditar las actividades de los Recursos Energéticos y Naturales No Renovables (ARCERNNR, s.f.).

Combustible a las embarcaciones que utilizan Gasolina de Pesca Artesanal para su Actividad" (Anexo 1) por parte de las gasolineras autorizadas.

De acuerdo con el reglamento impuesto por el organismos de control, es el deber de las gasolineras autorizadas evaluar si el cliente cumple con los requisitos antes de proceder a la venta de la gasolina para pesca artesanal, no obstante, dicho procedimiento se realiza de forma manual a expensas de los documentos originales del cliente, situación que puede generar inconsistencias a falta de una gestión de registros de compras organizado y una ineficiente comunicación entre distribuidores a lo largo del perfil costanero.

Por lo descrito, el desarrollo de un sistema de información centralizado dirigido a este proceso apoyará eficientemente al manejo de los datos a través de medios digitales, contribuyendo en gran medida al control que realizan los organismos pertinentes e impactando en la eficiencia de los distribuidores de la costa ecuatoriana al acatar el reglamento con el apoyo de herramientas digitales de uso diario.

1.4. OBJETIVOS

1.1.1. General:

Desarrollar un sistema de información prototipo para para el registro y control del despacho de gasolina de pesca artesanal, caso de estudio “Distribuidora del Puerto”.

1.1.2. Específicos:

1. Seleccionar una metodología ágil de desarrollo de software que se adapte a las necesidades del proyecto.
2. Definir los requerimientos funcionales y no funcionales del sistema de información para el caso de uso.
3. Elaborar el diseño de la aplicación conforme a la metodología ágil escogida, produciendo los documentos y entregables requeridos.
4. Desarrollar un sistema de información prototipo que cumpla los requerimientos funcionales.
5. Realizar pruebas de funcionalidad de la aplicación.

1.5. ALCANCE

El presente proyecto de titulación culminará con el desarrollo de un sistema de información a nivel prototipo para el registro y control del despacho de gasolina de pesca artesanal, para la “Distribuidora del Puerto, que es el caso de estudio. El sistema contará con interfaces de usuario en un entorno web, implementando las siguientes funcionalidades:

- Gestión de una base de datos para el almacenamiento y manipulación de la información de los clientes.
- Registro de ventas de la gasolina de pesca artesanal.
- Desarrollo de reportes aplicando filtros y parámetros acorde a los requerimientos funcionales.

1.6. LIMITACIONES

- La ausencia de un servicio de archivos para la manipulación directa de reportes, optando por una visualización previa del producto final previo a la interacción para descargar el archivo.
- La ausencia de un proceso de recuperación de contraseña, una decisión deliberada para mantener la simplicidad del sistema y optimizar el rendimiento dentro del alcance definido del prototipo.

CAPÍTULO II: MARCO TEÓRICO

2.1. SISTEMAS DE INFORMACIÓN

Son conjuntos integrados de componentes diseñados para la recolección, almacenamiento y procesamiento de datos, que juegan un papel crucial en la gestión operativa y estratégica de las organizaciones modernas. Facilitan una amplia gama de operaciones empresariales y funcionan como una columna vertebral en la interacción con clientes y proveedores, así como en la administración interna. Los principales componentes de estos sistemas incluyen el hardware y software informático, las telecomunicaciones y las bases de datos, todos ellos fundamentales para el funcionamiento eficiente de cualquier entidad. Estos sistemas permiten que las organizaciones manejen de manera eficaz grandes volúmenes de información y apoyen procesos críticos desde operaciones financieras hasta la gestión de recursos humanos (Zwass, 2023).

A medida que las tecnologías de la información han avanzado, también lo han hecho las capacidades de los sistemas de información, afectando profundamente la estructura organizacional y las estrategias de mercado. Estos sistemas no solo mejoran la eficiencia operativa, sino que también facilitan la innovación mediante la integración y análisis de datos. El desarrollo de redes más extensas y la implementación de almacenamiento en la nube han permitido que los recursos de los sistemas de información sean más accesibles y flexibles, ofreciendo así nuevas oportunidades para la colaboración y la expansión a nuevos mercados (Zwass, 2023). El impacto de los sistemas de información en la competitividad y productividad de las organizaciones es significativo, ya que permiten una gestión más efectiva de las operaciones.

2.1.1. Aplicación web

Las aplicaciones web son programas informáticos almacenados en servidores remotos y accesibles a través de navegadores web. Esta modalidad de software ofrece ventajas significativas debido a su compatibilidad con la mayoría de los sistemas operativos y computadoras estándar, sin requerir almacenamiento local en el disco duro del dispositivo. Además, las aplicaciones web son accesibles desde prácticamente cualquier dispositivo con conexión a internet, permitiendo la participación simultánea de múltiples

usuarios. Aunque dependen de una conexión de red, este requisito se ha vuelto menos restrictivo con la creciente ubicuidad de Internet.

En cuanto a su desarrollo, las aplicaciones web han evolucionado desde simples programas estáticos hasta complejos sistemas interactivos. Originalmente, cada página web en Internet era un documento estático y cualquier interacción requería que el servidor enviara una nueva página completa. Sin embargo, con la introducción de JavaScript en 1995 por Netscape Communications Corp., y posteriormente AJAX en 2005, se permitió a los desarrolladores mejorar significativamente la interactividad sin recargar completamente la página web. Estos avances permitieron una experiencia de usuario más rápida y agradable, lo que llevó a un aumento en la funcionalidad y popularidad de las aplicaciones web. La capacidad de estas aplicaciones se ha incrementado aún más con el lanzamiento de HTML5 en 2008, que introdujo mejoras significativas en el lenguaje de programación y en las interfaces de programación de aplicaciones (APIs) web, facilitando aún más el trabajo de los desarrolladores y mejorando la funcionalidad para los usuarios finales (Volle, 2022).

2.1.2. Frontend

El desarrollo front-end se enfoca en la parte visible de un sitio web, donde los usuarios interactúan directamente. Involucra la creación de menús desplegables, diseños y estilos a través del uso de lenguajes de programación como JavaScript, HTML y CSS. HTML establece la estructura y el contenido del sitio, mientras que CSS se encarga de los aspectos visuales como colores y diseño. JavaScript añade funciones interactivas avanzadas (Simmons, 2023). Los desarrolladores front-end también utilizan frameworks como jQuery y Bootstrap, y herramientas de diseño gráfico para optimizar la apariencia y la funcionalidad de las páginas web.

2.1.3. Backend

El desarrollo back-end se centra en el servidor, donde se maneja la lógica y la estructura que soporta el front-end. Los desarrolladores de back-end utilizan lenguajes de programación como Java, Python y Ruby para construir la funcionalidad del sitio, manejar bases de datos y asegurar la correcta ejecución de las aplicaciones a través de APIs. También trabajan con herramientas como SQL Server para almacenar y gestionar datos de

manera eficiente (Simmons, 2023). Este tipo de desarrollo es crucial para garantizar que el sitio web funcione correctamente y pueda responder a las solicitudes de los usuarios.

2.1.4. Fullstack

El desarrollo full-stack engloba tanto el front-end como el back-end de un sitio web, permitiendo a los desarrolladores trabajar en todas las etapas del proceso de creación de aplicaciones web. Estos desarrolladores tienen habilidades en lenguajes de programación utilizados tanto en el cliente como en el servidor, como JavaScript, Java, Python y CSS, y están versados en arquitectura web y desarrollo de algoritmos (Simmons, 2023). El dominio de full-stack permite una comprensión completa del flujo de datos y la interacción entre el servidor y el cliente, facilitando la implementación de soluciones integradas y cohesivas.

2.2. CICLO DE VIDA DE DASARROLLO DE SOFTWARE

El Ciclo de Vida de Desarrollo de Software (SDLC) es una estructura esencial para la planificación, creación y mantenimiento de software. Este marco define las fases críticas por las que un proyecto de software debe pasar, desde la conceptualización inicial hasta su finalización y mantenimiento. Cada fase del SDLC tiene objetivos específicos y actividades clave que aseguran que el software final cumpla con los requisitos de calidad y funcionalidad deseados (Al-Saqqa, 2020). A continuación, se describen detalladamente las fases típicas del SDLC.

2.2.1. Recolección de Requisitos

Esta fase inicial es fundamental para el éxito de cualquier proyecto de software. Consiste en identificar claramente las necesidades y expectativas de los usuarios finales y otros stakeholders. Durante esta etapa, se realizan entrevistas, encuestas y reuniones de revisión para recopilar todos los requisitos necesarios. La documentación detallada de estos requisitos es crucial, ya que sirve como una guía para todas las fases subsiguientes del desarrollo.

2.2.2. Diseño

En la fase de diseño, los requisitos recogidos se transforman en una especificación de diseño que será la base para el desarrollo del software. Se utilizan diagramas de casos de uso y documentación de diseño para visualizar la arquitectura y las interfaces del sistema. Esta fase asegura que la solución propuesta cumpla tanto con los requisitos técnicos como con los del negocio, y define cómo se estructurará y construirá el software.

2.2.3. Desarrollo

Durante la fase de desarrollo, el equipo convierte los diseños en un producto de software funcional. Se escribe código fuente basado en las especificaciones de diseño y se realizan revisiones y pruebas unitarias para garantizar que cada parte del software funcione correctamente. Esta fase requiere una colaboración intensiva entre los desarrolladores para asegurar cohesión y calidad en el código desarrollado.

2.2.4. Pruebas

Una vez que el software está desarrollado, entra en la fase de pruebas, donde se verifica exhaustivamente para detectar errores y defectos. Se prueban tanto las funcionalidades individuales como el sistema en su conjunto para asegurar que cumple con los requisitos establecidos. Esta fase es crítica para garantizar la estabilidad y funcionalidad del software antes de su despliegue.

2.2.5. Mantenimiento

La última fase del SDLC es el mantenimiento, que comienza una vez que el software está en operación. Incluye la realización de correcciones, mejoras y actualizaciones basadas en el feedback de los usuarios y cambios en el entorno tecnológico o empresarial. El mantenimiento asegura que el software continúe funcionando de manera eficaz y eficiente a lo largo del tiempo.

2.3. MÉTODOLOGÍAS DE DASARROLLO

El desarrollo de software es un campo dinámico y complejo que exige una gran adaptabilidad y precisión metodológica para abordar eficazmente los desafíos de la creación de soluciones tecnológicas. Este campo se caracteriza por la diversidad de enfoques metodológicos que se pueden clasificar en dos categorías principales: metodologías tradicionales y ágiles. Estas categorías no solo se diferencian en sus procesos y técnicas, sino que también se adaptan a diferentes tipos de proyectos, contextos operativos y objetivos estratégicos (Acharya & Kumar Sahu, 2020). A través de estos enfoques, se proporcionan marcos de trabajo específicos que guían todas las etapas dicho proceso, desde la concepción inicial hasta la entrega final del producto de software.

2.3.1. Metodologías tradicionales

Las metodologías tradicionales, a menudo referidas como enfoques en cascada, se basan en un proceso secuencial y lineal donde cada fase del proyecto fluye hacia la siguiente. Este enfoque estructurado es especialmente útil en proyectos con requisitos claramente definidos y donde los cambios son pocos o gestionables de manera predecible. Los proyectos que requieren una alta fiabilidad, como los sistemas bancarios, aplicaciones médicas o software aeroespacial, a menudo optan por metodologías tradicionales debido a su enfoque disciplinado en la planificación y la documentación exhaustiva, lo que ayuda a garantizar la precisión y minimizar los riesgos (Acharya & Kumar Sahu, 2020). Algunas de las metodologías tradicionales más populares incluyen el Modelo en Cascada, el Modelo V y el Modelo de Desarrollo en Espiral.

2.3.2. Metodologías ágiles

En contraste, las metodologías ágiles ofrecen un marco dinámico que promueve la iteración rápida, la flexibilidad y la adaptabilidad continua ante los cambios. Este enfoque es adecuado para proyectos en entornos volátiles donde los requisitos del cliente pueden evolucionar durante el proyecto, como es común en el desarrollo de software comercial y de consumo. Las metodologías ágiles fomentan una colaboración estrecha entre el equipo de desarrollo y los clientes, permitiendo ajustes regulares al producto que maximizan el valor entregado en cada iteración. Empresas de tecnología y startups, que necesitan lanzar

productos rápidamente y adaptarse a las respuestas del mercado, se benefician enormemente de la flexibilidad que ofrecen los enfoques ágiles (Acharya & Kumar Sahu, 2020). Algunas de las metodologías ágiles más populares incluyen Scrum, Programación Extrema (XP), Desarrollo Dirigido por Características (FDD) y el Método Kanban.

2.3.3. Comparación entre metodologías ágiles y tradicionales

El desarrollo de software se beneficia de la aplicación de diversas metodologías, cada una adaptada a diferentes escenarios y necesidades del proyecto. El siguiente cuadro comparativo entre metodologías tradicionales y ágiles, subraya las diferencias fundamentales en varios aspectos críticos del desarrollo.

Tabla 1

Cuadro comparativo entre metodologías ágiles y tradicionales

Característica	**Metodología Tradicional**	**Metodología Ágil**
Enfoque	Secuencial y lineal, cada fase fluye hacia la siguiente.	Dinámico, promueve iteración rápida y flexibilidad.
Adecuación del Proyecto	Proyectos con requisitos claramente definidos y cambios predecibles.	Proyectos en entornos volátiles con requisitos evolutivos.
Planificación	Detallada y disciplinada, importante en la fase inicial.	Flexible y adaptativa, continua a lo largo del proyecto.
Documentación	Extensa, para garantizar precisión y minimizar riesgos.	Menos enfatizada, enfocada en la rapidez y adaptabilidad.
Entrega	Al final del proyecto tras completar todas las fases.	Continua, en iteraciones regulares durante el proyecto
Enfoque	Secuencial y lineal, cada fase fluye hacia la siguiente.	Dinámico, promueve iteración rápida y flexibilidad.
Adecuación del Proyecto	Proyectos con requisitos claramente definidos y cambios predecibles.	Proyectos en entornos volátiles con requisitos evolutivos.
Planificación	Detallada y disciplinada, importante en la fase inicial.	Flexible y adaptativa, continua a lo largo del proyecto.
Documentación	Extensa, para garantizar precisión y minimizar riesgos.	Menos enfatizada, enfocada en la rapidez y adaptabilidad.

2.4. Metodología seleccionada: Extreme Programing

La elección de la metodología Extreme Programing, o XP, para este proyecto de desarrollo de software es ideal porque se busca un enfoque altamente adaptable. XP enfatiza la comunicación constante, la retroalimentación rápida, y el desarrollo iterativo, lo que resulta útil en proyectos que requieren cambios rápidos y flexibilidad (Agile Alliance, 2023). Su adopción puede ser particularmente efectiva en equipos pequeños y dinámicos que trabajan en entornos de desarrollo rápidos y cambiantes.

2.4.1. Ciclo de desarrollo: Metodología Extreme Programing

El desarrollo de software utilizando la metodología XP se distingue por varias etapas clave que estructuran y guían la entrega eficaz del software. Estas etapas son diseñadas para maximizar la adaptabilidad y la eficiencia del equipo de desarrollo, asegurando que el producto final sea de alta calidad y cumpla con las expectativas del cliente (Agile Alliance, 2023). A continuación, se detallan los componentes principales de este proceso:

2.4.1.1. Planificación

En la fase inicial del desarrollo de software con XP, el cliente y el equipo de desarrollo se reúnen para discutir los requerimientos del proyecto. Durante esta etapa, la gestión desglosa los requerimientos del cliente en historias de usuario y selecciona herramientas de productividad para su seguimiento. El equipo de desarrollo elabora un cronograma para la entrega del proyecto y proporciona estimaciones para seguir los plazos de entrega. Los proyectos XP se rastrean a través de iteraciones de una a dos semanas, después de las cuales se realizan demostraciones al cliente (Banerjee, 2024).

2.4.1.2. Diseño

Esta es la fase de ideación del producto, donde los líderes del equipo de desarrollo se conectan con el cliente para idear y diseñar las características principales del producto. En esta etapa, se mantiene deliberadamente baja la complejidad y las redundancias del proyecto para reducir el tiempo necesario para construir el producto y acelerar el tiempo de

salida al mercado (Banerjee, 2024). Los diseñadores e ingenieros del proyecto también tienen la tarea de asegurar que la lógica del diseño sea simple y fácil de seguir.

2.4.1.3. Desarrollo y pruebas

En la etapa de desarrollo, los programadores crean el código del proyecto de manera estructurada, siguiendo convenciones estándar. Utilizan la programación en pareja, donde dos programadores colaboran en el mismo módulo: uno escribe el código y el otro lo revisa, corrige errores y sugiere mejoras (Banerjee, 2024). Todo el equipo tiene acceso a los documentos de diseño y puede contribuir en cualquier parte del proyecto, manteniendo una propiedad colectiva del código.

En la fase de pruebas, se da gran importancia a la retroalimentación del cliente. Los tests de aceptación, también conocidos como pruebas funcionales o pruebas del cliente, son esenciales en este proceso. Estas pruebas validan la visión del cliente sobre la aplicación y se diferencian de las pruebas unitarias, que verifican la perspectiva interna del software según el programador. Los programadores ayudan a redactar los tests de aceptación, pero la participación directa del cliente es crucial para garantizar que las expectativas del cliente se cumplan plenamente (Nagler, 2017). Cada prueba codifica uno o más escenarios específicos de usuario y simula automáticamente las acciones del usuario, validando los resultados esperados para asegurar que el producto final cumpla con las expectativas del cliente.

2.4.2. Entregables de la Metodología Extreme Programing

La metodología XP destaca por su enfoque minimalista y práctico, que se centra en la documentación esencial y efectiva para el desarrollo y mantenimiento del software (Agile Alliance, 2023). A continuación, se describen los principales tipos de documentación y entregables generados en un proyecto de XP:

2.4.2.1. Historias de usuario

En XP, la documentación de requerimientos se maneja de manera directa y personalizada. Se utiliza una comunicación verbal intensiva con el cliente, quien permanece 'in situ' con el equipo. Esto facilita un entendimiento claro y directo de las necesidades sin la necesidad de extensos documentos escritos. Las historias de usuario se

capturan en tarjetas escritas y son apoyadas por documentación complementaria como tablas de valores o extractos de documentos de requerimientos externos (Jeffries, 2001).

2.4.2.2. Documentación de diseño

El diseño en XP crece de manera incremental, lo cual reduce la necesidad de documentación extensa. Los artefactos de diseño suelen ser efímeros; por ejemplo, se pueden dibujar diagramas UML en una pizarra y luego construir directamente la funcionalidad. Es común que los equipos de XP mantengan algunas representaciones visuales del diseño del sistema, pero estas sirven más como referencia ocasional que como documentación formal (Jeffries, 2001).

2.5. HERRAMIENTAS DE DASARROLLO

Las herramientas de desarrollo de software son programas que utilizan los desarrolladores para crear, mantener y mejorar otras aplicaciones y programas. Estas herramientas, como editores de código, compiladores y depuradores son esenciales en el campo de la tecnología porque ayudan a los desarrolladores a realizar su trabajo de manera eficiente y precisa (GoodFirms, 2024). Al igual que un carpintero necesita martillos y sierras, un desarrollador necesita herramientas adecuadas para construir software.

Las herramientas de desarrollo a utilizar en este proyecto de titulación son PlantUML para el diseño UML, NextJS para el desarrollo del sistema web y PostgreSQL para el manejo de la base de datos.

2.5.1. PlantUML

PlantUML es una herramienta que permite graficar diagramas UML utilizando una descripción de texto simple y fácil de leer. Su uso en este proyecto se justifica por su enfoque basado en texto, que facilita la comprensión y edición de diagramas, y por ser de código abierto y gratuito, eliminando barreras económicas. Además, su algoritmo de diseño inteligente organiza los elementos de manera clara y eficiente, agilizando la creación y actualización de diagramas (PlantUML, 2009). La capacidad de personalización y el respaldo de una comunidad activa de usuarios hacen de PlantUML una opción versátil y poderosa para la representación gráfica en este proyecto.

2.5.2. NextJS

Next.js es un marco de trabajo diseñado para facilitar la construcción de aplicaciones web rápidas y completas, aprovechando las capacidades de React, una biblioteca enfocada en la creación de interfaces de usuario. Esta combinación permite a los desarrolladores trabajar con una base de herramientas y configuraciones preestablecidas, optimizando así el proceso de desarrollo (Vercel, Inc., 2024). Next.js mejora significativamente la eficiencia al resolver problemas comunes automáticamente, lo que permite a los desarrolladores concentrarse más en la personalización y mejora de sus aplicaciones.

2.5.3. PostgreSQL

PostgreSQL es un sistema de base de datos que ha sido desarrollado por más de 35 años y es conocido por su confiabilidad y seguridad. Funciona en todos los sistemas operativos principales y es gratuito y de código abierto, lo que significa que cualquiera puede usarlo y modificarlo. Además, PostgreSQL permite a los usuarios adaptar la base de datos a sus necesidades específicas, como agregar nuevas funciones o tipos de datos (The PostgreSQL Global Development Group, 2024). Es una opción popular para guardar y gestionar datos, grande o pequeños, de manera eficaz.

2.6. CASO DE USO

El sistema de información prototipo para el registro y control del despacho de gasolina de pesca artesanal, se desarrollará para la Distribuidora del Puerto, la cual es una distribuidora de Gasolina Pesca Artesanal (GPA), cuya única actividad es el despacho de la gasolina que utilizan los pescadores artesanales para realizar sus faenas de pesca a lo largo del perfil costanero de Esmeraldas. Está ubicada en la provincia de Esmeraldas, en el sector del Puerto Pesquero Artesanal, dirigido por Secretaría Técnica de Gestión Inmobiliaria del Sector Público (INMOBILIAR) que es una entidad del estado.

Dicho establecimiento funciona bajo la normativa de la Agencia de Regulación y Control de Energía y Recursos Naturales No Renovables (ARCERNNR), organismo de control de las gasolineras que distribuyen gasolina artesanal para la pesca, quienes se

encargan de la supervisión de las operaciones en el sector de hidrocarburos, asegurando el cumplimiento de las normativas y estándares establecidos. Sus responsabilidades abarcan desde la exploración y explotación de hidrocarburos hasta su transformación, transporte, almacenamiento y comercialización (INEC, s.f.). La ARCERNNR juega un papel crucial en la gestión de este sector, garantizando no solo la eficiencia y seguridad de las operaciones, sino también la protección del medio ambiente y los intereses de los consumidores y usuarios finales.

2.6.1. Situación actual

La Distribuidora del Puerto se adhiere estrictamente a los procedimientos de control establecidos por las autoridades gubernamentales. No obstante, en lo que respecta al registro y control de las ventas de gasolina para la pesca artesanal, se emplean cuadernos para el registro acumulativo de las transacciones diarias, y documentos en Excel para gestionar las bases de datos de clientes e insumos, cumpliendo así con los reportes exigidos por los organismos reguladores.

2.6.2. Normativas para las gasolineras autorizadas

La circular emitida por la ARCERNNR especifica que, bajo el Artículo 9 del Acuerdo Interinstitucional, el despacho de combustible a embarcaciones dedicadas a la pesca artesanal requiere presentación de documentos específicos. Estos documentos incluyen la cédula de ciudadanía del armador o de su delegado, una autorización formal si el retiro lo hace un delegado, y el documento de zarpe de la embarcación (véase Anexo 2), el cual es la autorización de pesca provista por la capitanía. Se enfatiza la importancia del cumplimiento de estas directrices, destacando la responsabilidad de los centros de distribución en la verificación y el seguimiento de estos requisitos. Además, señala que las estaciones de servicio que despachen combustible sin la debida autorización enfrentarán sanciones administrativas, y el transporte ilegal de combustibles puede resultar en penas de prisión (ARCERNNR, 2023).

CAPÍTULO III: REQUERIMIENTOS DEL SISTEMA

En este capítulo, abordaremos los requerimientos necesarios para el sistema que será implementado, agrupados en dos categorías esenciales: requisitos funcionales y no funcionales. Los requisitos funcionales se centran en las acciones específicas y el comportamiento deseado del sistema, dictando las funcionalidades esenciales que debe ofrecer. Por otro lado, los requisitos no funcionales especifican las condiciones bajo las cuales el sistema debe operar para garantizar una experiencia de usuario óptima y cumplir con los estándares operativos requeridos (Jama Software, 2023).

3.1. REQUERIMIENTOS FUNCIONALES

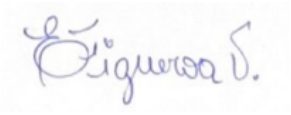
F1	Gestión de sesiones
F2	Gestión de roles de usuario
F3	Gestión de usuarios
F4	Gestión de categorías
F5	Gestión de parámetros
F6	Gestión de compras
F7	Gestión de personas
F8	Gestión de embarcaciones
F9	Gestión de motores
F10	Gestión de zarpes
F11	Gestión de ventas
F12	Gestión de reportes

3.2. HISTORIAS DE USUARIO

Tabla 2

Historia de usuario para el requerimiento funcional gestión de sesiones

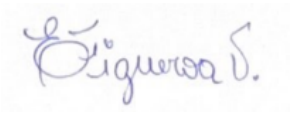
Historia de usuario	
Código de historia	1
Actores	Administrador, Usuario
Título de la historia	Gestión de sesiones
Prioridad en el negocio	Medio

Riesgo en el desarrollo	Alto
Número de iteración	1
Descripción	Este actor podrá ingresar al sistema utilizando credenciales para asegurar que solo usuarios y administradores autorizados tengan acceso a la información y funcionalidades del sistema.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 3

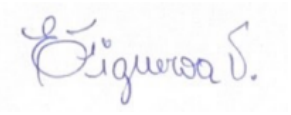
Historia de usuario para el requerimiento funcional gestión de roles de usuario

Historia de usuario	
Código de historia	2
Actores	Administrador
Título de la historia	Gestión de roles de usuario
Prioridad en el negocio	Medio
Riesgo en el desarrollo	Alto
Número de iteración	1
Descripción	Este actor podrá crear, leer, modificar y eliminar roles de usuario, asignando los permisos correspondientes en los diferentes módulos del sistema.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 4

Historia de usuario para el requerimiento funcional gestión de usuarios

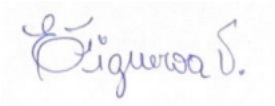
Historia de usuario	
Código de historia	3
Actores	Administrador
Título de la historia	Gestión de usuarios
Prioridad en el negocio	Medio
Riesgo en el desarrollo	Alto
Número de iteración	1
Descripción	Este actor podrá crear, leer, modificar y eliminar usuarios del sistema, asignándoles roles específicos, para asegurar que cada usuario tenga los permisos adecuados y el acceso controlado a las funcionalidades del sistema.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 5

Historia de usuario para el requerimiento funcional gestión de categorías

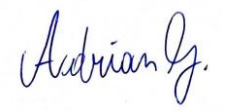
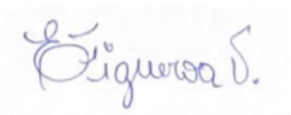
Historia de usuario	
Código de historia	4
Actores	Administrador
Título de la historia	Gestión de categorías
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Medio
Número de iteración	2

Descripción	Este actor podrá crear, leer, modificar y eliminar categorías para designar características a los registros de determinados módulos del sistema.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 6

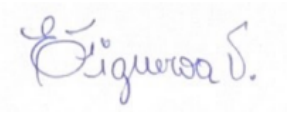
Historia de usuario para el requerimiento funcional gestión de parámetros

Historia de usuario	
Código de historia	5
Actores	Administrador
Título de la historia	Gestión de parámetros
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Medio
Número de iteración	2
Descripción	Este actor podrá gestionar los parámetros del sistema, específicamente el precio y el IVA, para asegurar que los cálculos realizados sean precisos y puedan ajustarse fácilmente en caso de cambios futuros.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 7

Historia de usuario para el requerimiento funcional gestión de compras

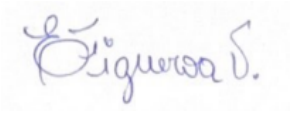
Historia de usuario	
Código de historia	6
Actores	Administrador
Título de la historia	Gestión de compras
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Medio
Número de iteración	2
Descripción	Este actor podrá crear, leer, modificar y eliminar registros de compras en el sistema, incluyendo detalles como el número de factura, la fecha de la compra, la cantidad de galones y el respectivo valor pagado.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 8

Historia de usuario para el requerimiento funcional gestión de personas

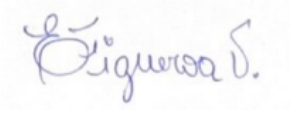
Historia de usuario	
Código de historia	7
Actores	Usuario
Título de la historia	Gestión de personas
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Bajo
Número de iteración	3
Descripción	Este actor podrá crear, leer, modificar y eliminar registros de personas en el sistema, para gestionar de manera centralizada tanto a clientes como a propietarios y tripulantes, evitando la necesidad de módulos separados para cada tipo de persona.

Firma del desarrollador	
	Adrián Galeas Figueroa
Firma del cliente	
	Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 9

Historia de usuario para el requerimiento funcional gestión de embarcaciones

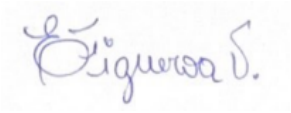
Historia de usuario	
Código de historia	8
Actores	Usuario
Título de la historia	Gestión de embarcaciones
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Bajo
Número de iteración	3
Descripción	Este actor podrá crear, leer, modificar y eliminar registros de embarcaciones en el sistema, incluyendo a las personas en virtud de propietario y tripulante, quienes pueden ser diferentes o una persona misma.
Firma del desarrollador	
	Adrián Galeas Figueroa
Firma del cliente	
	Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 10

Historia de usuario para el requerimiento funcional gestión de motores

Historia de usuario

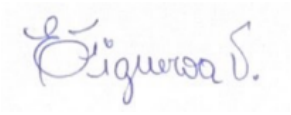
Código de historia	9
Actores	Usuario
Título de la historia	Gestión de motores
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Bajo
Número de iteración	3
Descripción	Este actor podrá crear, leer, modificar y eliminar registros de motores en el sistema, asignándolos a su embarcación correspondiente y designando su categoría.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 11

Historia de usuario para el requerimiento funcional gestión de zarpes

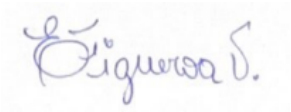
Historia de usuario	
Código de historia	10
Actores	Usuario
Título de la historia	Gestión de zarpes
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Bajo
Número de iteración	3
Descripción	Este actor podrá crear, leer, modificar y eliminar registros de zarpes en el sistema, asignándolos a las embarcaciones correspondientes y especificando su fecha de expedición y caducidad.
Firma del desarrollador	 Adrián Galeas Figueroa

Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 12

Historia de usuario para el requerimiento funcional gestión de ventas

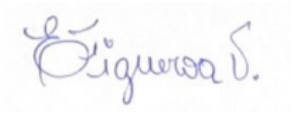
Historia de usuario	
Código de historia	11
Actores	Usuario
Título de la historia	Gestión de ventas
Prioridad en el negocio	Alto
Riesgo en el desarrollo	Alto
Número de iteración	4
Descripción	Este actor podrá crear, leer, modificar y eliminar registros de ventas en el sistema, incluyendo detalles como la fecha de la venta, la cantidad de galones, la embarcación asociada, el cliente, el método de pago y el respectivo valor pagado.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

Tabla 13

Historia de usuario para el requerimiento funcional gestión de reportes

Historia de usuario	
Código de historia	12
Actores	Usuario
Título de la historia	Gestión de reportes
Prioridad en el negocio	Alto

Riesgo en el desarrollo	Alto
Número de iteración	4
Descripción	Este actor podrá visualizar los reportes en el sistema, aplicar y manipular filtros para personalizar la información, y luego descargar los reportes, para obtener datos relevantes y específicos de manera eficiente y precisa.
Firma del desarrollador	 Adrián Galeas Figueroa
Firma del cliente	 Elsy Figueroa Villegas

Fuente: Producción propia

3.3. REQUERIMIENTOS NO FUNCIONALES

NF1 Seguridad de Datos

El sistema protege la información utilizando credenciales de acceso y asignando permisos específicos basados en los roles de los usuarios, lo que asegura que solo las personas autorizadas puedan acceder a datos sensibles.

NF2 Integridad de Datos

El sistema mantiene la integridad de los datos implementando controles estrictos en la entrada de información, asegurando que los datos ingresados en cada campo de las entidades sean precisos y estén bien validados.

NF3 Rendimiento

El sistema optimiza el rendimiento al ubicar tanto el front-end como el back-end en el mismo entorno de hosting, lo que minimiza la latencia y maximiza la velocidad de respuesta. Esta configuración integrada también simplifica la administración y el mantenimiento del sistema, resultando en una gestión más eficiente de los recursos computacionales.

CAPÍTULO IV: DISEÑO

4.1. DIAGRAMAS DE CASO DE USO GENERAL

En este capítulo se presenta el diagrama de caso de uso a nivel general, proporcionando una vista de alto nivel de las funcionalidades del sistema y las interacciones principales entre los actores y los casos de uso.

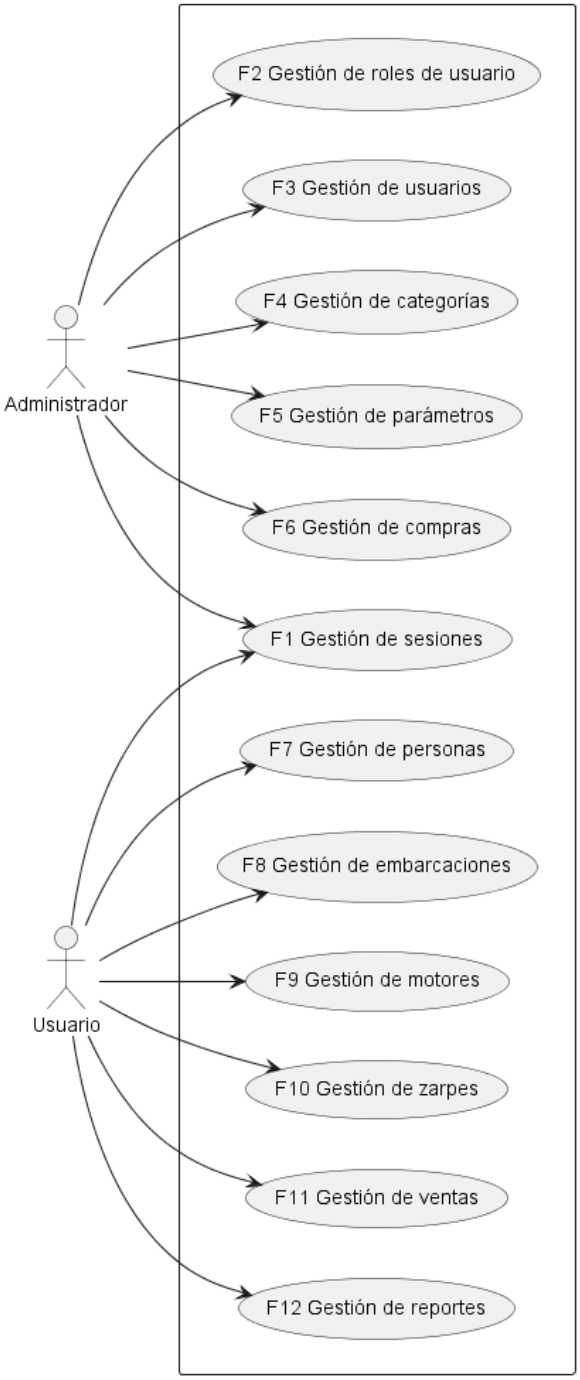


Figura 1. Diagrama de caso de uso general de este proyecto.

4.2. DIAGRAMAS DE CASO DE USO A DETALLE

En este capítulo se presentan los diagramas de caso de uso correspondientes a los requerimientos funcionales, mostrando una visión general de cada caso de uso. Estos diagramas ilustran cómo interactúan los diferentes componentes del sistema, proporcionando una comprensión clara de las funcionalidades sin entrar en un desglose detallado de cada paso.

4.2.1. Caso a detalle para el requerimiento funcional Gestión de sesiones

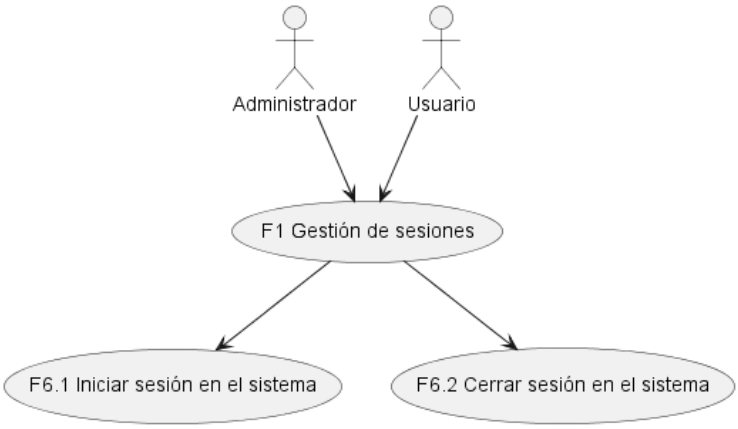


Figura 2. Diagrama de caso de uso para el requerimiento funcional F1 Gestión de sesiones.

4.2.2. Caso a detalle para el requerimiento funcional Gestión de roles de usuario

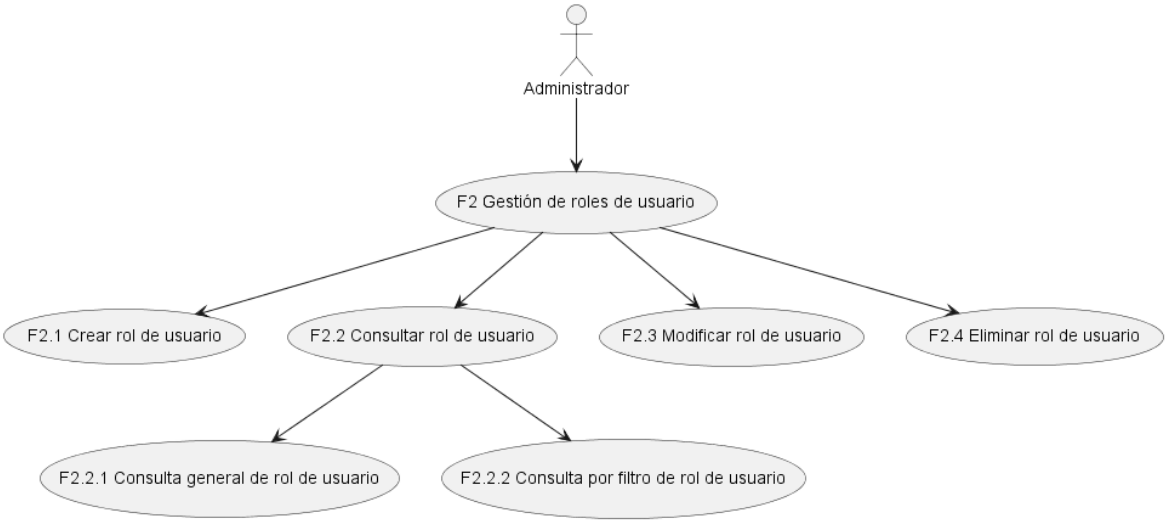


Figura 3. Diagrama de caso de uso para el requerimiento funcional F2 Gestión de roles de usuario.

4.2.3. Caso a detalle para el requerimiento funcional Gestión de usuarios

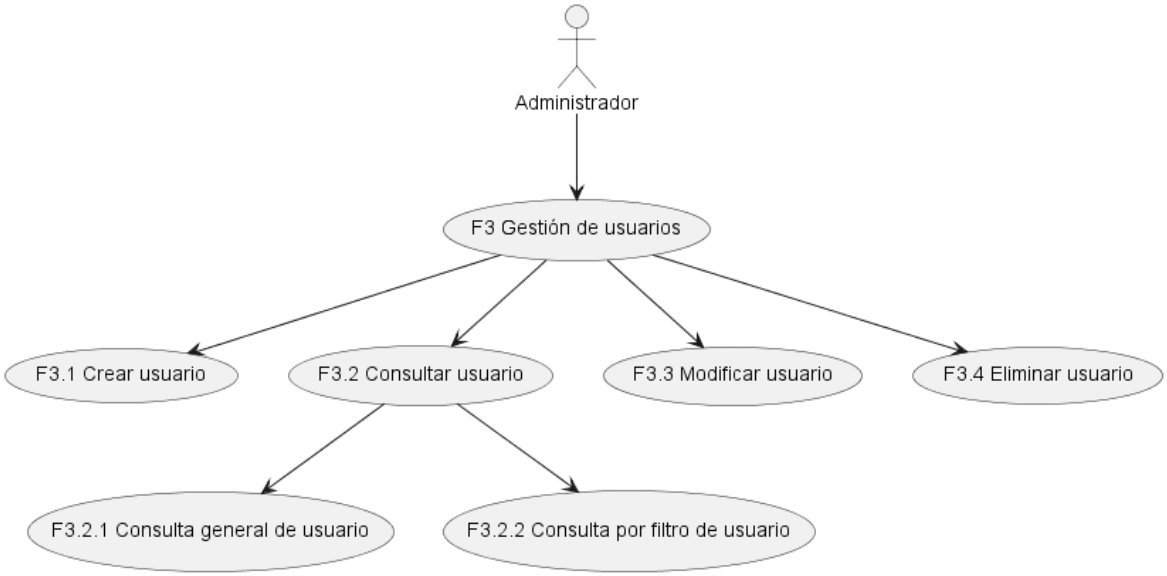


Figura 4. Diagrama de caso de uso para el requerimiento funcional F3 Gestión de usuarios.

4.2.4. Caso a detalle para el requerimiento funcional Gestión de categorías

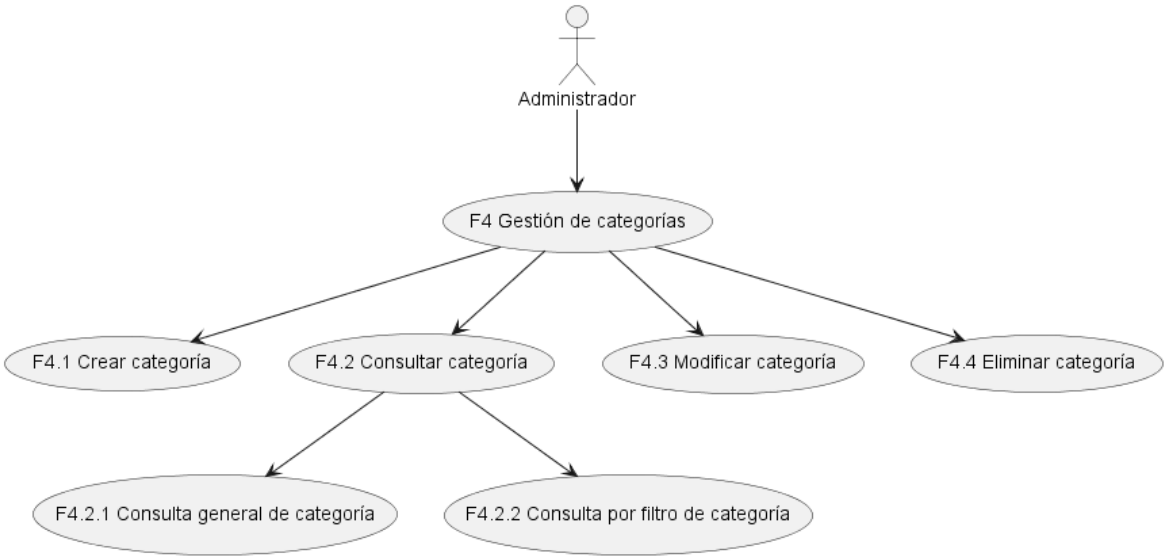


Figura 5. Diagrama de caso de uso para el requerimiento funcional F4 Gestión de categorías.

4.2.5. Caso a detalle para el requerimiento funcional Gestión de parámetros

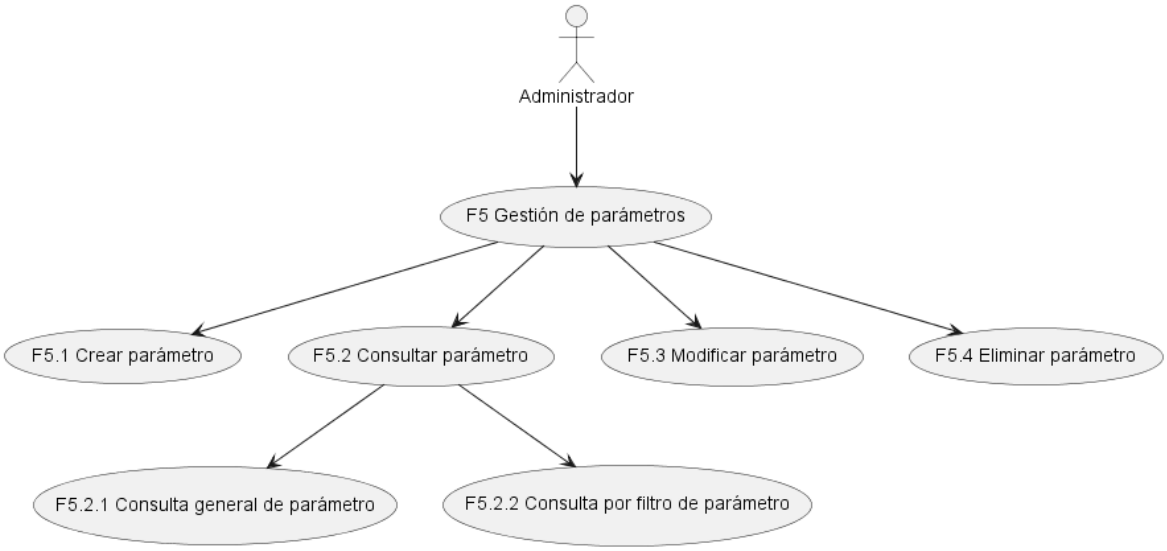


Figura 6. Diagrama de caso de uso para el requerimiento funcional F5 Gestión de parámetros.

4.2.6. Caso de uso para el requerimiento funcional Gestión de compras

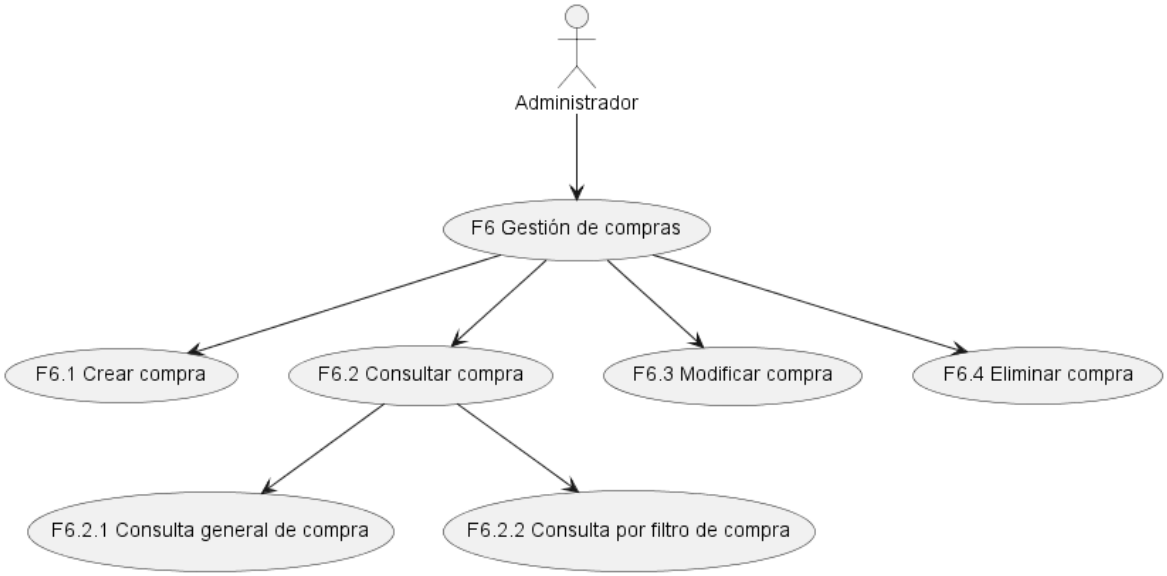


Figura 7. Diagrama de caso de uso para el requerimiento funcional F6 Gestión de compras.

4.2.7. Caso a detalle para el requerimiento funcional Gestión de personas

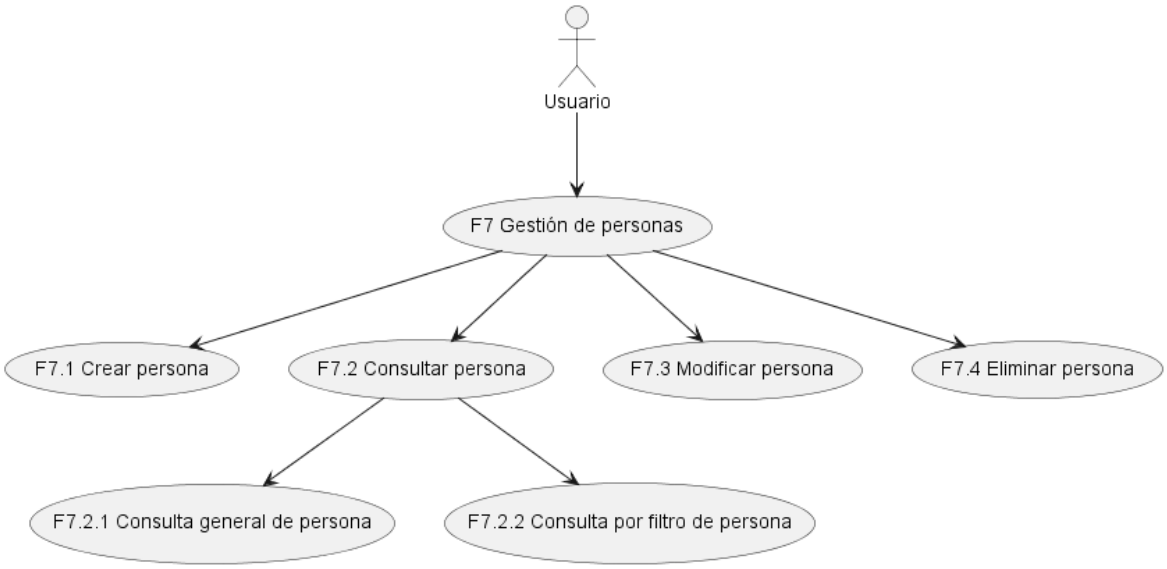


Figura 8. Diagrama de caso de uso para el requerimiento funcional F7 Gestión de personas.

4.2.8. Caso a detalle para el requerimiento funcional Gestión de embarcaciones

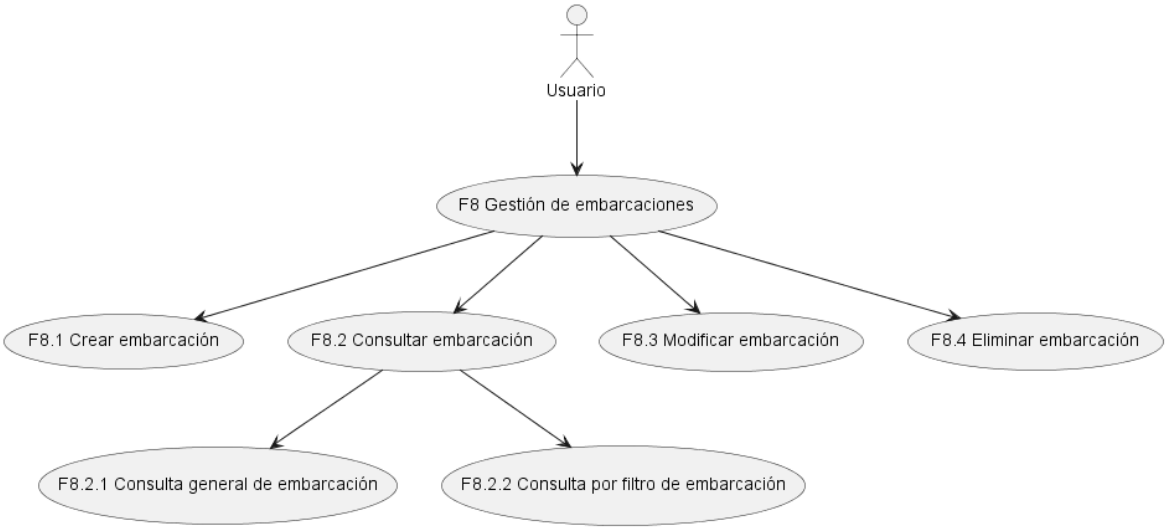


Figura 9. Diagrama de caso de uso para el requerimiento funcional F8 Gestión de embarcaciones.

4.2.9. Caso a detalle para el requerimiento funcional Gestión de motores

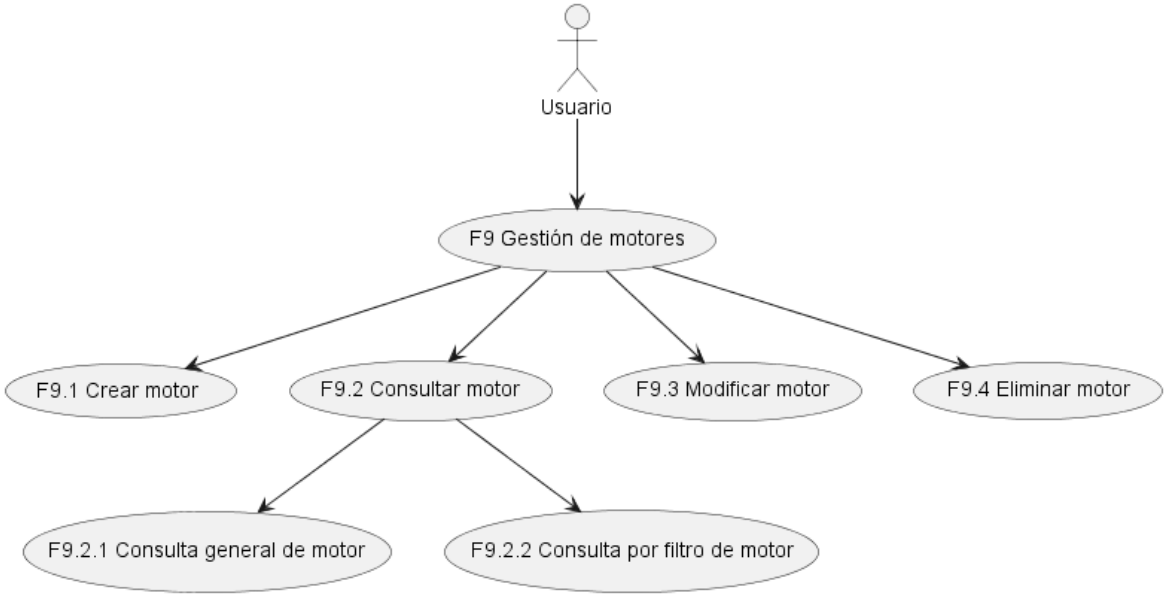


Figura 10. Diagrama de caso de uso para el requerimiento funcional F9 Gestión de motores.

4.2.10. Caso a detalle para el requerimiento funcional Gestión de zarpes

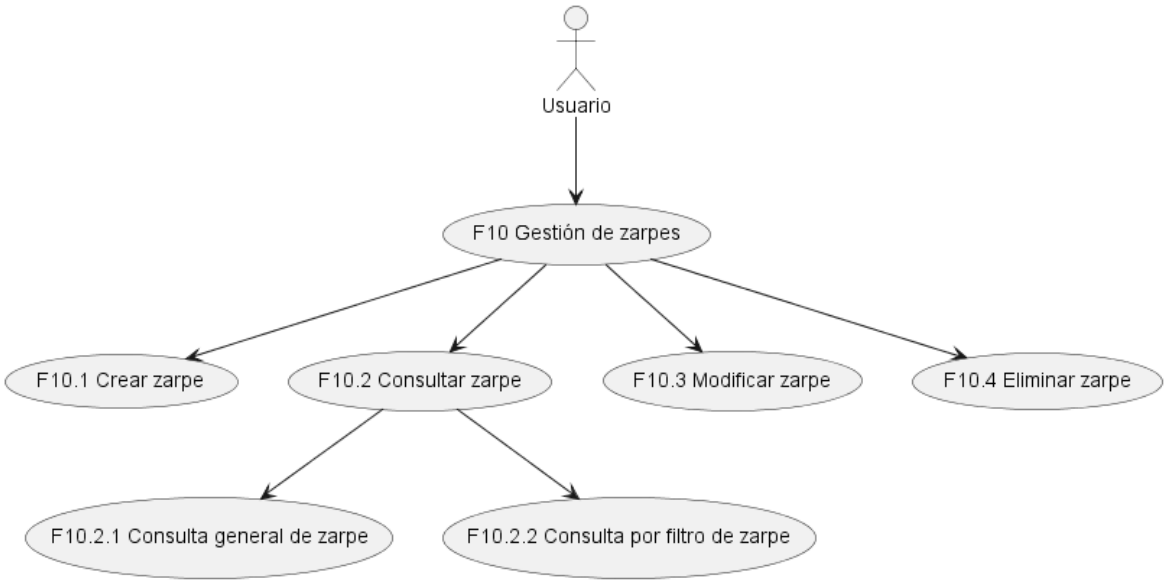


Figura 11. Diagrama de caso de uso para el requerimiento funcional F10 Gestión de zarpes.

4.2.11. Caso a detalle para el requerimiento funcional Gestión de ventas

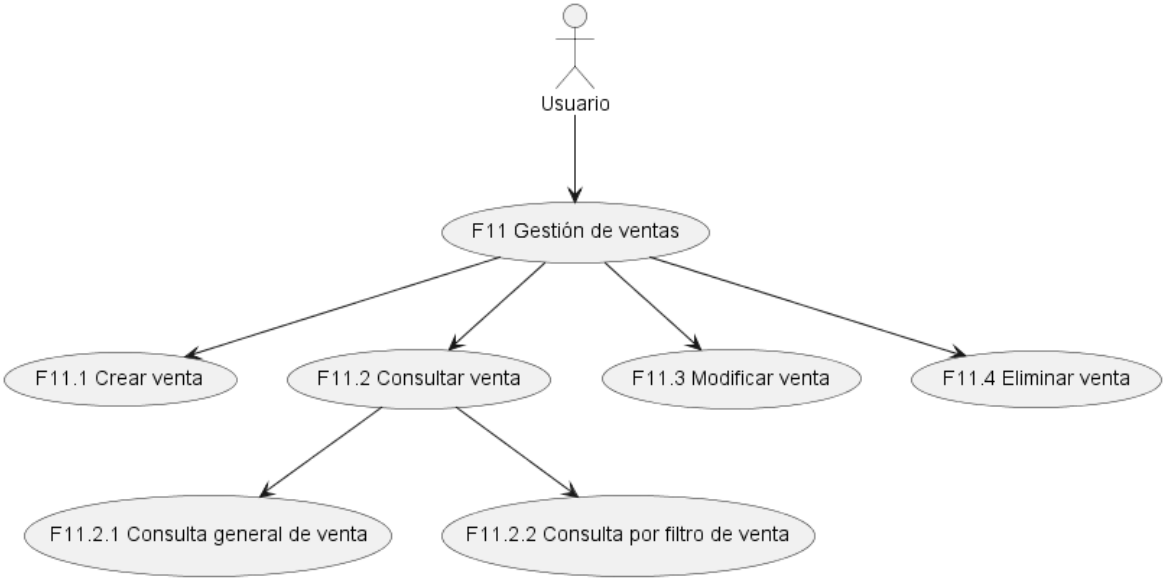


Figura 12. Diagrama de caso de uso para el requerimiento funcional F11 Gestión de ventas.

4.2.12. Caso a detalle para el requerimiento funcional Gestión de reportes

Figura 13. Diagrama de caso de uso para el requerimiento funcional F12 Gestión de reportes.

4.3. DISEÑO DE LA BASE DE DATOS

El diseño de la base de datos es un proceso esencial que garantiza la organización y eficiencia del almacenamiento de información en un sistema. Este diseño se desarrolla en tres etapas: conceptual, lógico y físico, cada una aportando diferentes niveles de detalle y especificidad.

4.3.1. Diseño conceptual de la base de datos

El diseño conceptual se enfoca en definir las entidades principales y las relaciones entre ellas. En esta etapa, se identifican los elementos fundamentales que se gestionarán en la base de datos, sin entrar en detalles técnicos (Taylor, 2023). El objetivo es crear un modelo abstracto que represente la estructura general de la información.

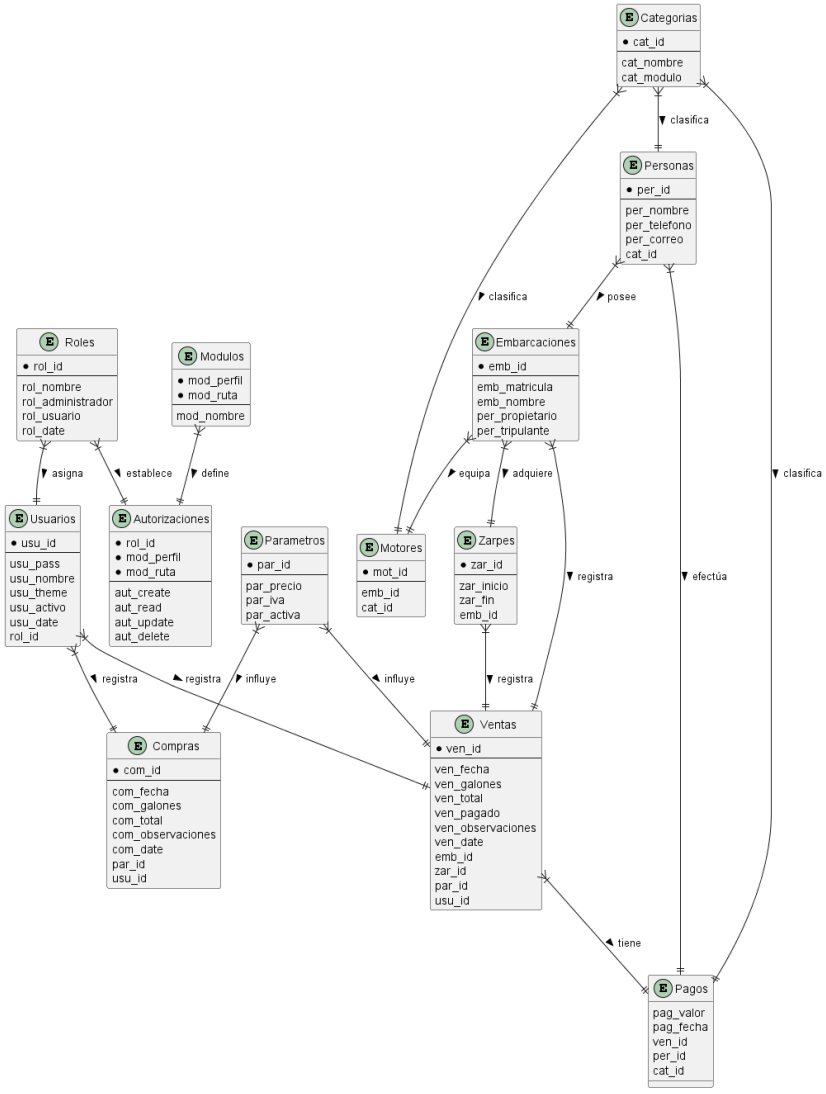


Figura 14. Diseño conceptual de la base de datos utilizando PlantUML.

4.3.2. Diseño lógico de la base de datos

El diseño lógico detalla las estructuras de datos y sus atributos, especificando los tipos de datos y las relaciones normalizadas (Taylor, 2023). En esta fase, se desarrollan modelos más precisos, asegurando que la información esté organizada de manera eficiente y coherente, sin redundancias innecesarias.

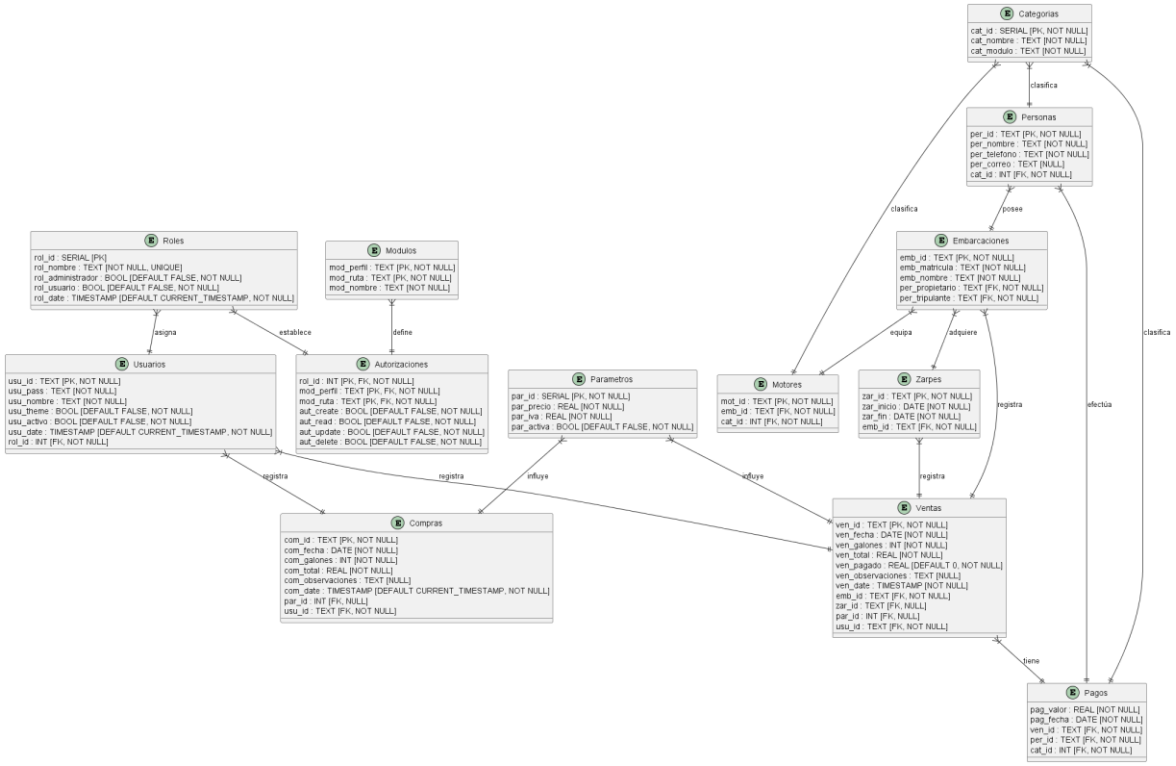


Figura 15. Diseño lógico de la base de datos utilizando PlantUML.

4.3.3. Diseño físico de la base de datos

El diseño físico aborda la implementación real de la base de datos en un sistema específico. Esta etapa incluye la definición de índices, tipos de datos específicos del sistema de gestión de bases de datos (SGBD) utilizado, y la disposición física de los datos en el almacenamiento (Taylor, 2023). El objetivo es optimizar el rendimiento y la integridad de la base de datos.

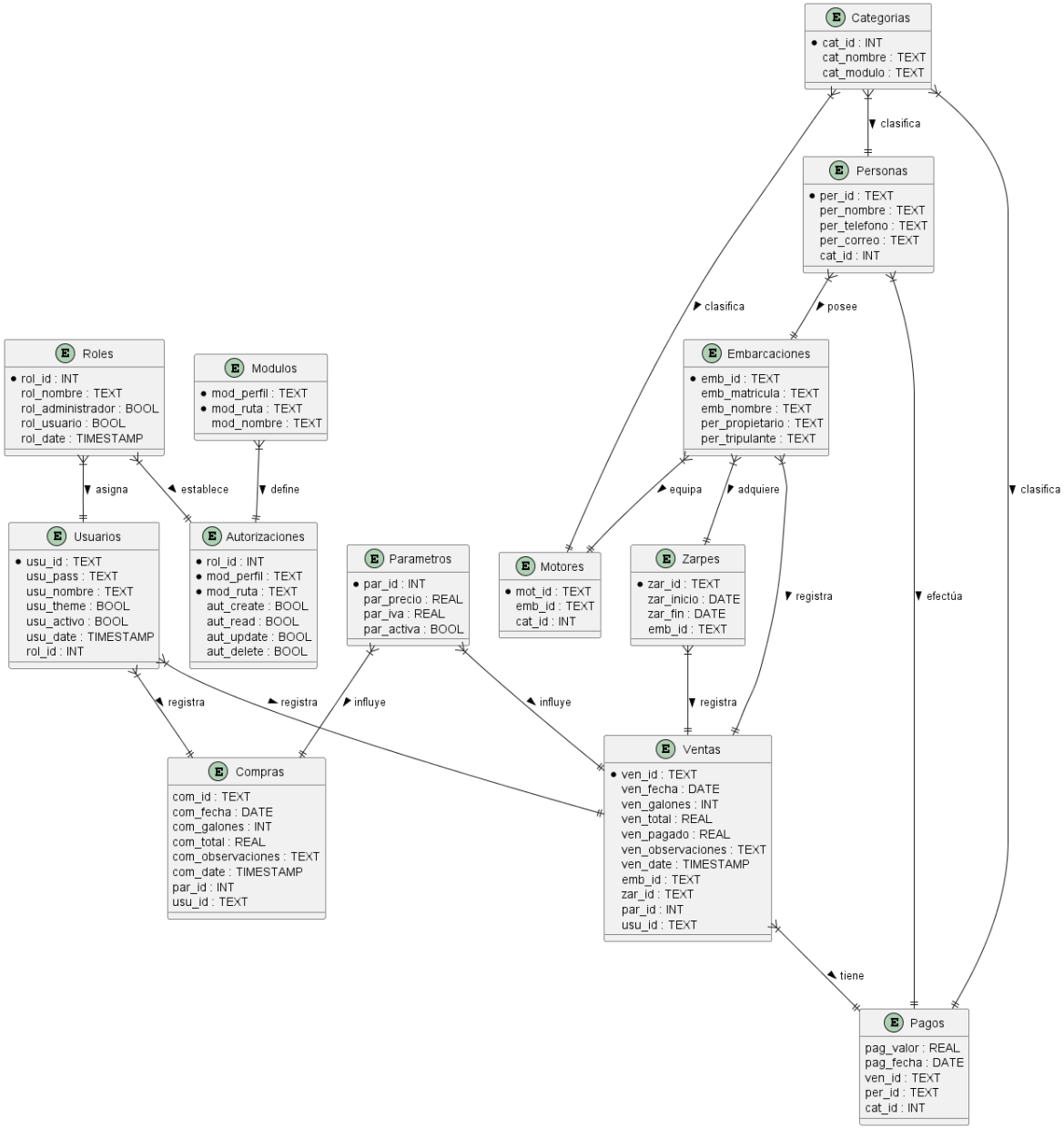


Figura 16. Diseño físico de la base de datos utilizando PlantUML.

CAPÍTULO V: DESARROLLO Y PRUEBAS

En este capítulo se detallará el proceso de desarrollo y pruebas del proyecto, destacando las metodologías y prácticas utilizadas para garantizar la calidad y alineación con los requisitos del cliente. El desarrollo se llevó a cabo en iteraciones, cada una de las cuales incluye la programación y las pruebas de aceptación correspondientes. Los ajustes se realizaron en base a los comentarios recibidos.

5.1. ITERACIONES

En este capítulo se describen las cuatro iteraciones planificadas para el desarrollo del sistema, cada una con tareas específicas asignadas para asegurar un progreso continuo y organizado. La división en iteraciones permite manejar el riesgo y la complejidad del proyecto, asegurando que las funcionalidades críticas sean implementadas primero y que se puedan realizar ajustes sobre la marcha. Esta planificación también facilita la gestión del tiempo y los recursos, garantizando la entrega de un sistema funcional y de alta calidad.

Tabla 14

Planificación de iteraciones en torno a prioridad, riesgo y justificación

Iteración	Prioridad	Riesgo	Justificación
1	Medio	Alto	Implementar estas características permite una gestión detallada de roles y usuarios, asegurando un control adecuado de acceso al sistema.
2	Alto	Medio	Estas funcionalidades son esenciales para establecer la gestión de compras, categorías y parámetros, permitiendo la organización inicial del sistema.
3	Alto	Bajo	Estas funcionalidades amplían la capacidad del sistema para manejar entidades clave, como personas, embarcaciones, motores y zarpes, asegurando una gestión integral.
4	Alto	Alto	Estas características finales son cruciales para el funcionamiento operativo del sistema, permitiendo la gestión de ventas y la generación de reportes para el análisis y la toma de decisiones.

Fuente: Producción propia

5.2. CRONOGRAMA

En este capítulo se presenta el cronograma del proyecto, detallando las fechas de inicio y finalización de cada iteración planificada. Este cronograma se ha elaborado para asegurar que el proyecto se complete dentro del plazo establecido, permitiendo un seguimiento claro del progreso y facilitando la gestión de tiempos y recursos. La estructura de las iteraciones está diseñada para abordar primero las funcionalidades más críticas y luego las complementarias, garantizando un desarrollo coherente y organizado.

Tabla 15

Planificación de iteraciones en torno duración, progreso y prioridad

Iteración	Fecha de Inicio	Fecha de Vencimiento	Prioridad
Iteración 1	8/4/2024	21/4/2024	Media
Iteración 2	22/4/2024	5/5/2024	Importante
Iteración 3	6/5/2024	19/5/2024	Importante
Iteración 4	20/5/2024	2/6/2024	Importante

Fuente: Producción propia

Cada iteración incluye tareas específicas asignadas al programador responsable, Adrián Galeas Figueroa. El progreso de cada iteración se monitoriza de manera continua para asegurar que se cumplan los plazos y se mantenga la calidad del desarrollo. Las fechas de vencimiento establecidas proporcionan un marco temporal claro para la finalización de cada fase del proyecto, permitiendo ajustes oportunos si es necesario.

5.3. HERRAMIENTAS DE SEGUIMIENTO

Para el seguimiento y la gestión de las tareas del proyecto, se usará Microsoft Planner. Esta herramienta, parte de la suite de Microsoft 365, permite organizar, asignar y monitorizar tareas de manera efectiva. Con Microsoft Planner, se pueden crear planes, asignar tareas y establecer fechas de vencimiento (Stsepanets, 2023).

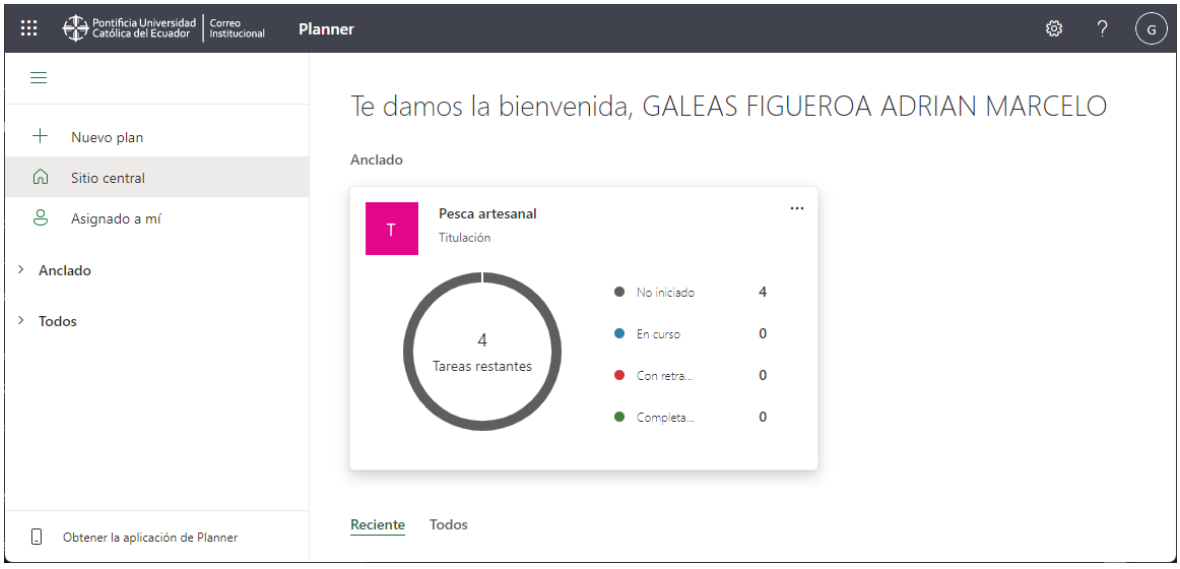


Figura 17. Pantalla principal de Microsoft Planner con el plan de este proyecto configurado.

En este proyecto basado en la metodología XP, donde la iteración y la respuesta rápida a los cambios son esenciales, el uso de Microsoft Planner contribuye a una mayor eficiencia y organización al centralizar todas las tareas en una plataforma, permitiendo un seguimiento ágil y efectivo de todas las actividades del proyecto.

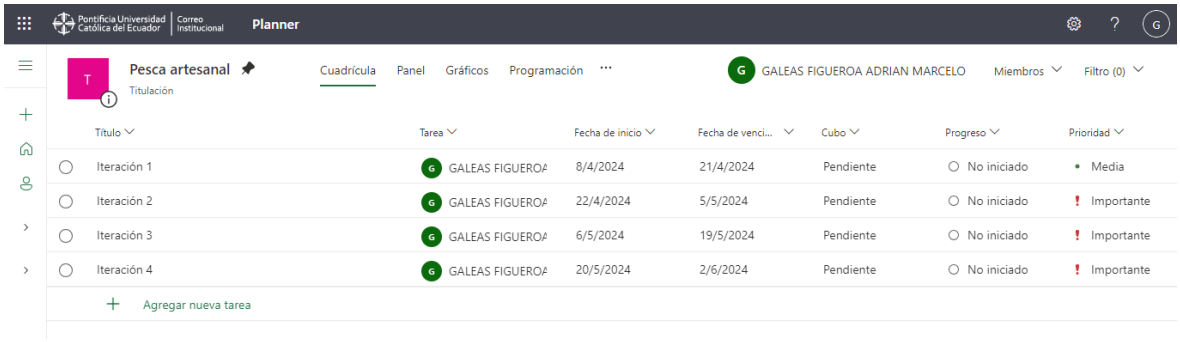


Figura 18. Cuadrícula del plan de este proyecto de titulación en Microsoft Planner.

5.4. ITERACIÓN 1

Durante esta iteración, se implementaron las funcionalidades básicas del sistema, centrándose en la gestión de usuarios y roles. El objetivo principal era establecer una base sólida para la gestión de sesiones del sistema.

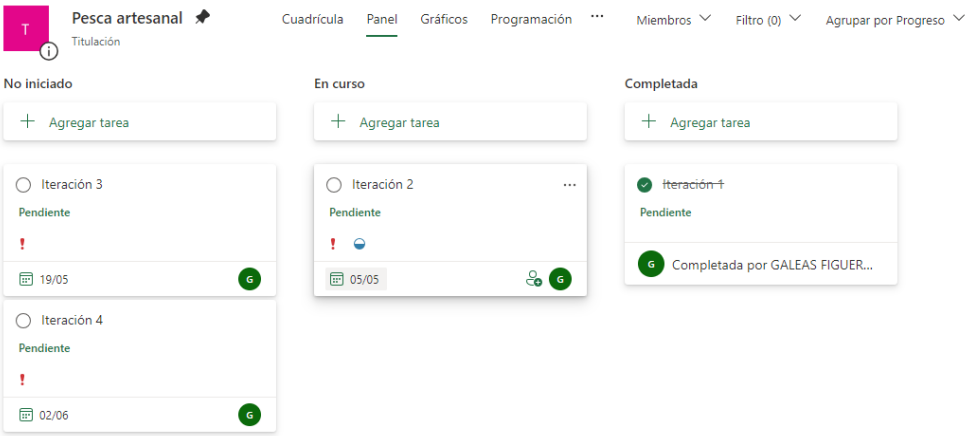


Figura 19. Primera actualización del cronograma basado en el progreso de las iteraciones.

5.4.1. F1 Gestión de sesiones

La funcionalidad de gestión de sesiones es crucial para asegurar que sólo los usuarios autorizados accedan al sistema. Esta sección del sistema permite a los usuarios del sistema iniciar y cerrar sesión de manera segura. El código para esta funcionalidad se encuentra en el Anexo 3.

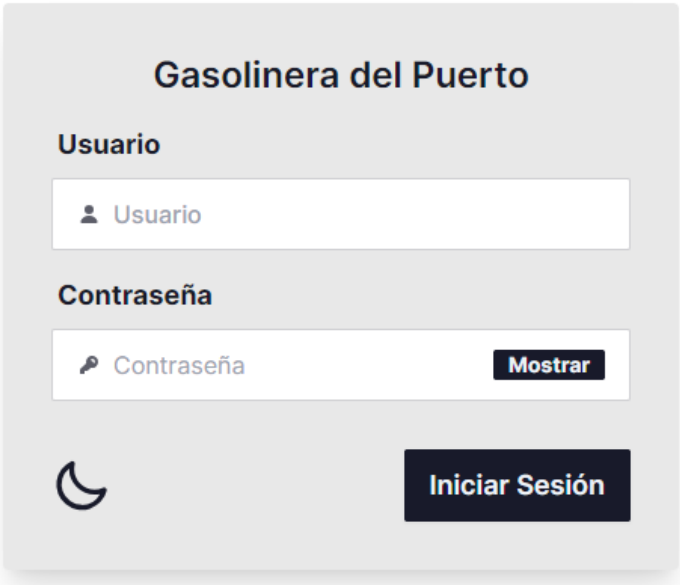


Figura 20. Interfaz de usuario para F1.1 Ingresar al sistema.

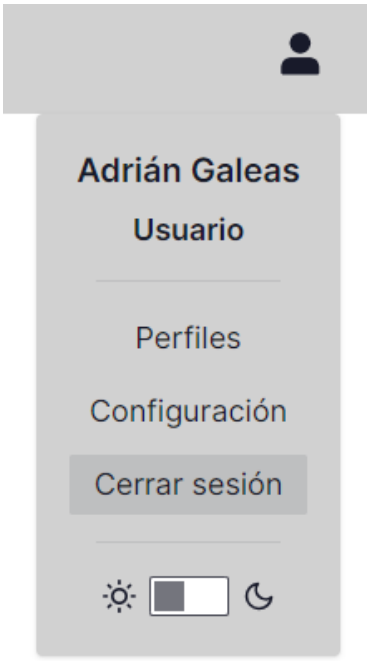
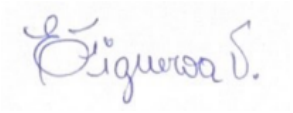


Figura 21. Interfaz de usuario para F1.2 Salir del sistema.

Tabla 16

Pruebas de usuario para el requerimiento Gestión de sesiones

	Caso de prueba
Código de prueba	1
Actores	Administrador, Usuario
Historia de usuario	1. Gestión de sesiones
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Llenar el formulario con su código de usuario y contraseña. 3. Pulsar el botón “Iniciar sesión” 4. Dar clic en el ícono superior derecho y desplegar el menú de opciones 5. Seleccionar la opción “Cerrar Sesión”
Resultado esperado	Ingreso y cierre de sesión en el sistema para acceder a las funcionalidades de este sin errores, reflejando los cambios correctamente y asegurando que el sistema gestione las sesiones de manera segura.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo tanto el inicio como el cierre de sesión correctamente. No se presentaron errores durante el proceso y las sesiones se manejaron de manera segura.

Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Las funcionalidades de gestión de sesiones operaron conforme a lo esperado, proporcionando un manejo seguro y efectivo de las sesiones de usuario, demostrando así la funcionalidad y la seguridad del sistema en operaciones críticas de autenticación y cierre de sesión.

Fuente: Producción propia

5.4.2. F2 Gestión de roles de usuario

Esta funcionalidad permite al administrador definir y gestionar los roles y permisos de los usuarios dentro del sistema. Esto facilita una distribución clara de responsabilidades y accesos dentro de la plataforma. El código para esta funcionalidad se encuentra en el Anexo 4.

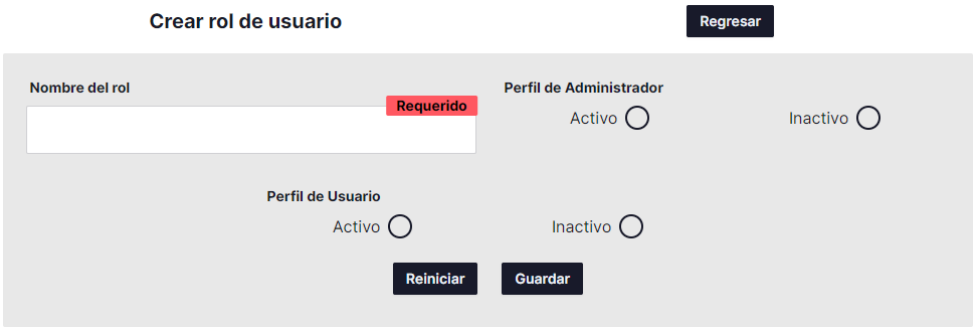


Figura 22. Interfaz de usuario para F2.1 Crear rol de usuario.

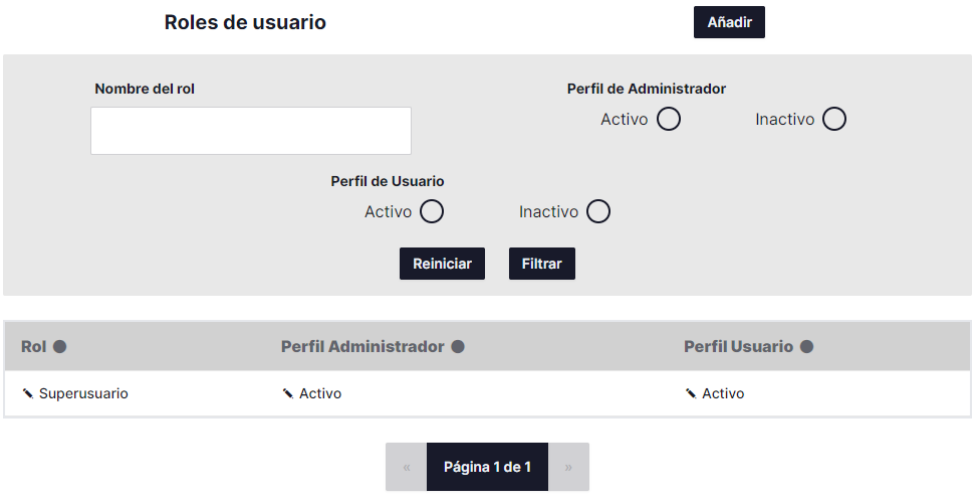


Figura 23. Interfaz de usuario para F2.2 Consultar roles de usuario.

Figura 24. Interfaz de usuario para F2.3 Modificar rol de usuario.

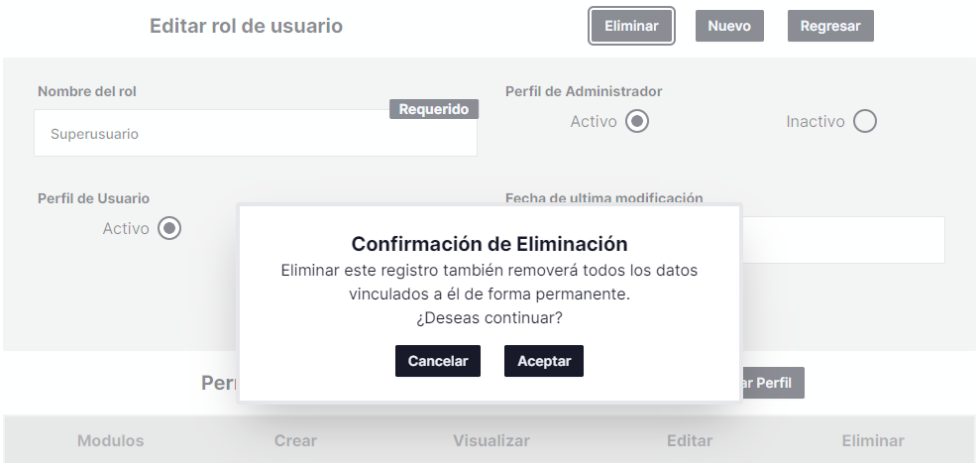
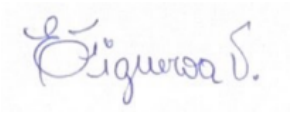


Figura 25. Interfaz de usuario para F2.4 Eliminar rol de usuario.

Tabla 17

Pruebas de usuario para el requerimiento Gestión de roles de usuario

Caso de prueba	
Código de prueba	2
Historia de usuario	2. Gestión de roles de usuario
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Sistema” de la barra de navegación superior. 3. Pulsar el botón “Roles de usuario” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar” 7. En la misma página modificar el rol creado anteriormente. 8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado. 10. Buscar el rol modificado anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 11. En la página del rol seleccionado hacer clic en el botón “Eliminar” del encabezado. 12. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar un rol de usuario sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar roles de usuario siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de roles de usuario. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de roles de usuario funcionaron como se esperaba, demostrando la integridad y eficacia

del sistema en la manipulación de datos de roles de usuario.

Fuente: Producción propia

5.4.3. F3 Gestión de usuarios

La gestión de usuarios permite al administrador controlar el acceso y las actividades de los usuarios en el sistema. El código para esta funcionalidad se encuentra en el Anexo 5.

Figura 26. Interfaz de usuario para F3.1 Crear usuario.

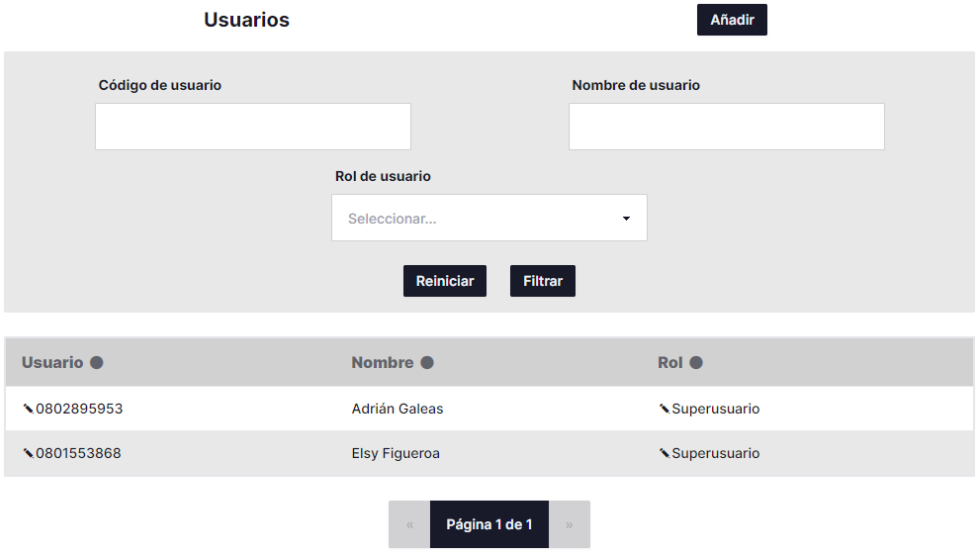


Figura 27. Interfaz de usuario para F3.2 Consultar usuarios.

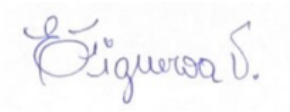
Figura 28. Interfaz de usuario para F3.3 Modificar usuario.

Figura 29. Interfaz de usuario para F3.4 Eliminar usuario.

Tabla 18

Pruebas de usuario para el requerimiento Gestión de usuarios

Caso de prueba	
Código de prueba	3
Historia de usuario	3. Gestión de usuarios
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Sistema” de la barra de navegación superior. 3. Pulsar el botón “Usuarios” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar” 7. En la misma página modificar el usuario creado anteriormente.

	8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado. 10. Buscar el usuario modificado anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 11. En la página del usuario seleccionado hacer clic en el botón “Eliminar” del encabezado. 12. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar un usuario sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar categorías siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de categorías. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de categorías funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de categorías.

Fuente: Producción propia

5.5. ITERACIÓN 2

En esta iteración, se añadieron funcionalidades para la gestión compras, categorías y parámetros del sistema, estableciendo parámetros críticos como el precio y el IVA.

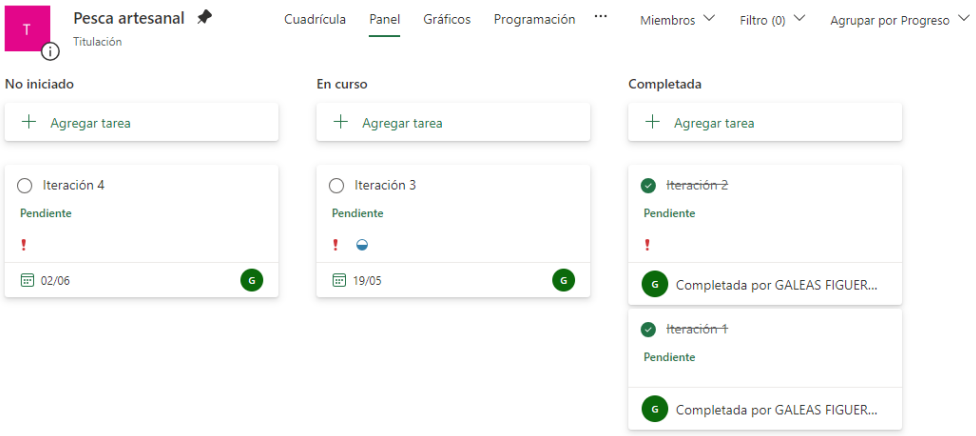


Figura 30. Segunda actualización del cronograma basado en el progreso de las iteraciones.

5.5.1. F4 Gestión de categorías

La funcionalidad de gestión de categorías permite al administrador crear, leer, modificar y eliminar categorías, lo cual facilita la organización de los registros en distintos módulos del sistema. El código para esta funcionalidad se encuentra en el Anexo 6.

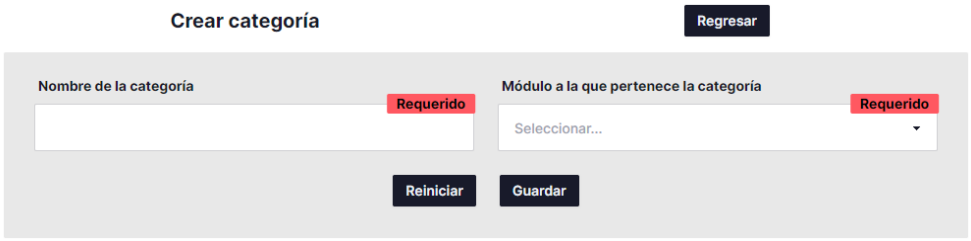


Figura 31. Interfaz de usuario para F4.1 Crear categoría.

Figura 32. Interfaz de usuario para F4.2 Consultar categoría.

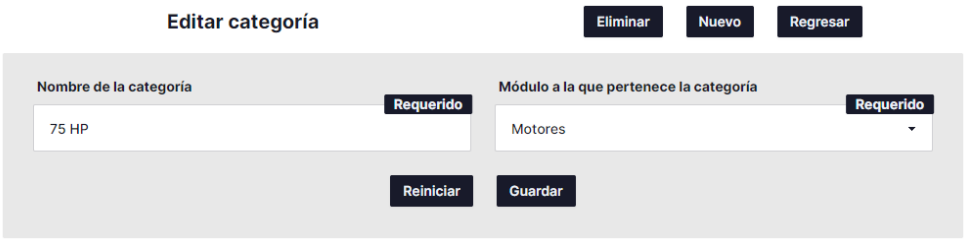


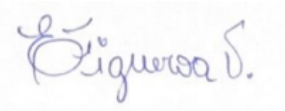
Figura 33. Interfaz de usuario para F4.3 Modificar categoría.

Figura 34. Interfaz de usuario para F4.4 Eliminar categoría.

Tabla 19

Pruebas de usuario para el requerimiento Gestión de categorías

Caso de prueba	
Código de prueba	4
Historia de usuario	4. Gestión de categorías
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Configuraciones” de la barra de navegación superior. 3. Pulsar el botón “Categorías” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar” 7. En la misma página modificar la categoría creada anteriormente. 8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado. 10. Buscar la categoría modificada anteriormente en la tabla inferior y

	seleccionarlo dando clic en el primer campo de la fila. 11. En la página de la categoría seleccionada hacer clic en el botón "Eliminar" del encabezado. 12. Leer la cláusula y dar clic en el botón "Aceptar".
Resultado esperado	El sistema permite agregar, modificar y eliminar una categoría sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar categorías siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de categorías. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de categorías funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de categorías.

Fuente: Producción propia

5.5.2. F5 Gestión de parámetros

Esta funcionalidad permite al administrador gestionar parámetros cruciales como el precio y el IVA, asegurando que los cálculos del sistema sean correctos y flexibles ante cambios. El código para esta funcionalidad se encuentra en el Anexo 7.

Figura 35. Interfaz de usuario para F5.1 Crear parámetro.

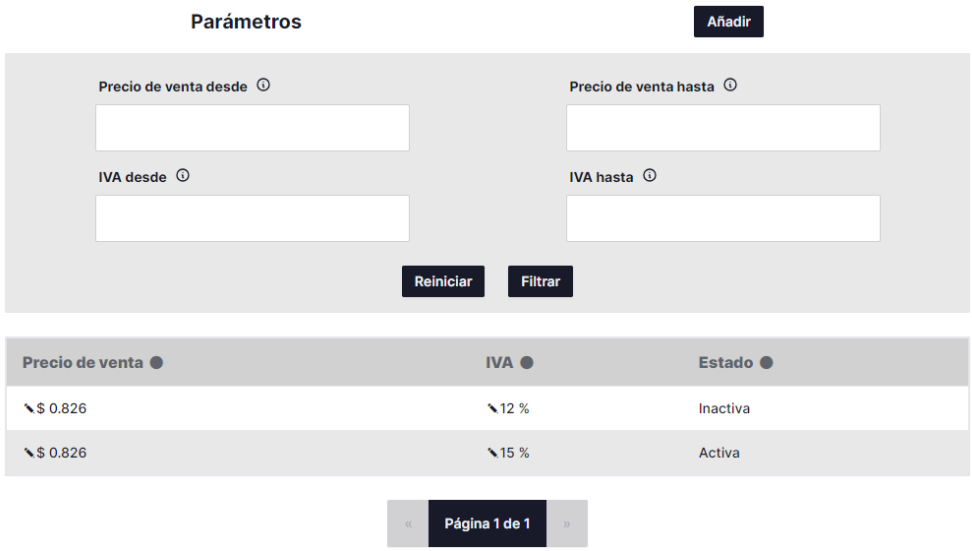


Figura 36. Interfaz de usuario para F5.2 Consultar parámetro.

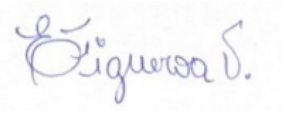
Figura 37. Interfaz de usuario para F5.3 Modificar parámetro.

Figura 38. Interfaz de usuario para F5.4 Eliminar parámetro.

Tabla 20

Pruebas de usuario para el requerimiento Gestión de parámetros

Caso de prueba	
Código de prueba	5
Historia de usuario	5. Gestión de parámetros
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Configuraciones” de la barra de navegación superior. 3. Pulsar el botón “Parámetros” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar” 7. En la misma página modificar el parámetro creado anteriormente. 8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado. 10. Buscar el parámetro modificado anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 11. En la página del parámetro seleccionado hacer clic en el botón “Eliminar” del encabezado. 12. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar un parámetro sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar parámetros siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación,

	modificación y eliminación de parámetros. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de parámetros funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de parámetros.

Fuente: Producción propia

5.5.3. F6 Gestión de compras

Permite al administrador gestionar eficazmente las compras, incluyendo la creación, modificación y eliminación de registros de compras, lo cual es vital para la administración financiera. El código para esta funcionalidad se encuentra en el Anexo 8.

Figura 39. Interfaz de usuario para F6.1 Crear compra.

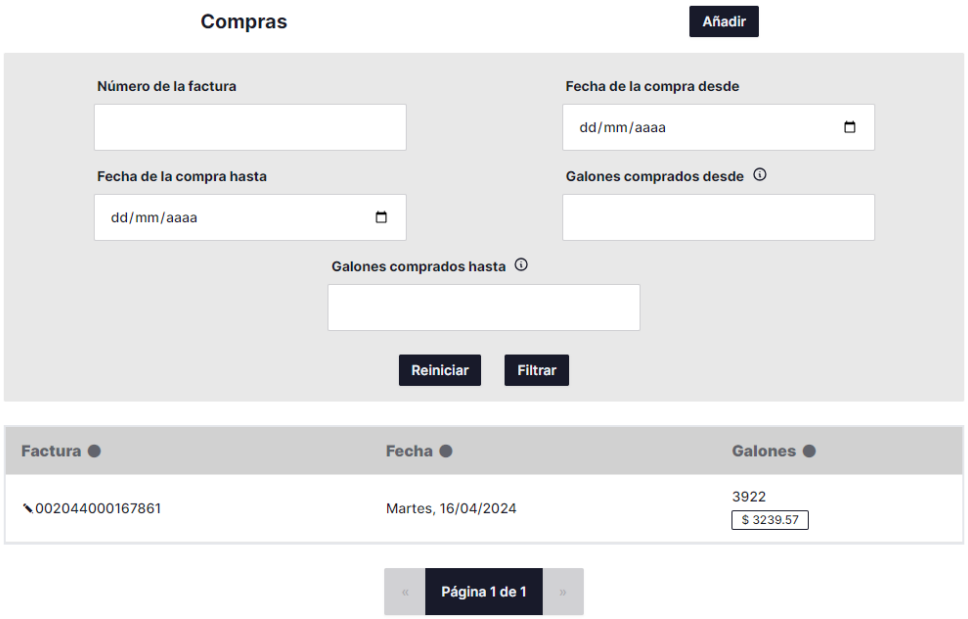


Figura 40. Interfaz de usuario para F6.2 Consultar compra.

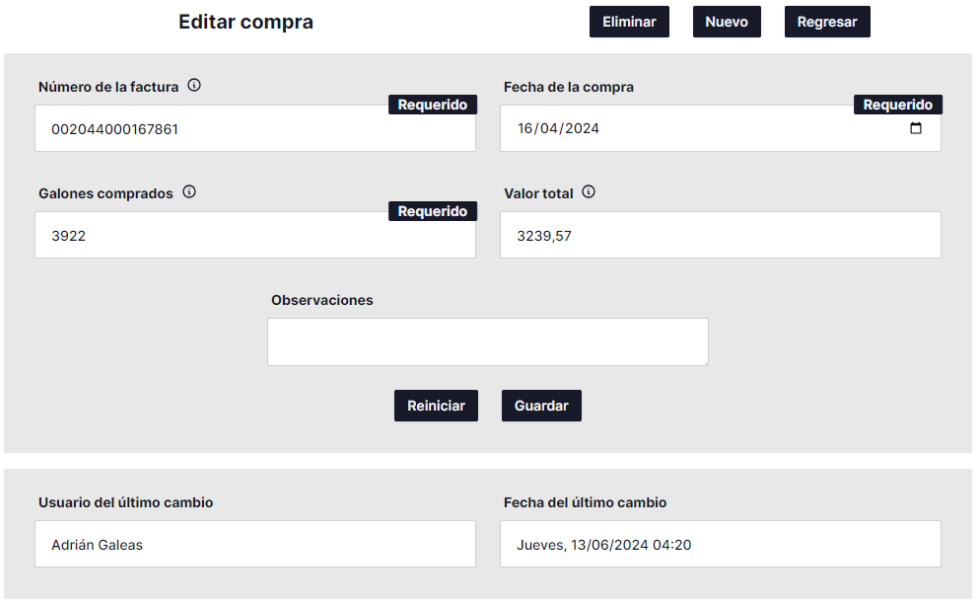


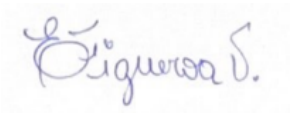
Figura 41. Interfaz de usuario para F6.3 Modificar compra.

Figura 42. Interfaz de usuario para F6.4 Eliminar compra.

Tabla 21

Pruebas de usuario para el requerimiento Gestión de roles de usuario

Caso de prueba	
Código de prueba	6
Historia de usuario	6. Gestión de compras
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Compras” de la barra de navegación superior. 3. Dar clic en el botón “Añadir” del encabezado de la página. 4. Completar la información de los campos requeridos. 5. Dar clic en el botón “Guardar” 6. En la misma página modificar la compra creada anteriormente. 7. Dar clic en el botón “Guardar” 8. En la misma página dar clic en el botón “Regresar” del encabezado. 9. Buscar la compra modificada anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 10. En la página de la compra seleccionado hacer clic en el botón “Eliminar” del encabezado. 11. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar una compra sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar compras siguiendo el flujo de acciones sin

	interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de compras. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de compras funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de compras.

Fuente: Producción propia

5.6. ITERACIÓN 3

En esta iteración, se implementaron funcionalidades para la gestión de personas, embarcaciones y motores, ampliando la capacidad del sistema para manejar entidades clave.

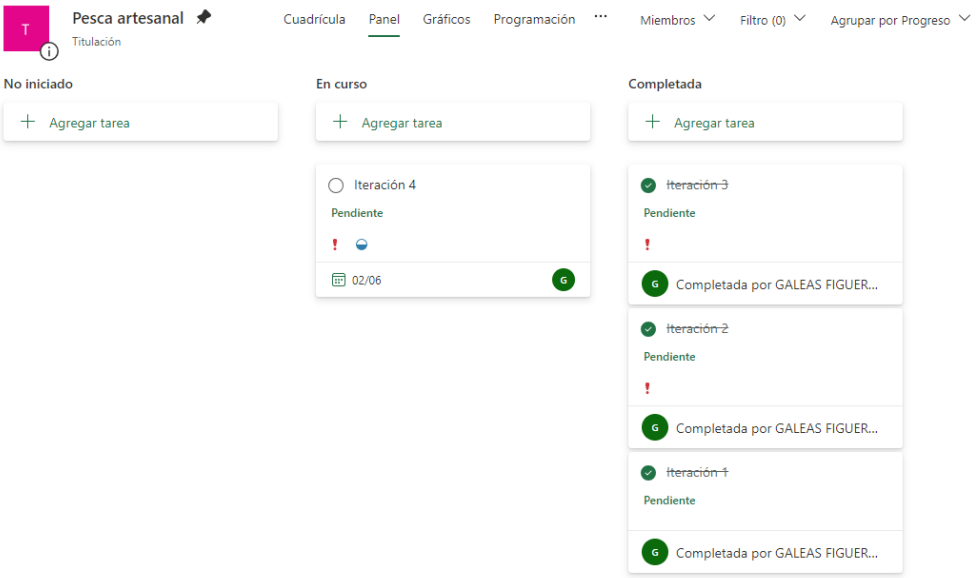


Figura 43. Tercera actualización del cronograma basado en el progreso de las iteraciones.

5.6.1. F7 Gestión de personas

Esta funcionalidad centraliza la gestión de información de clientes, facilitando la administración y evitando duplicidades. El código para esta funcionalidad se encuentra en el Anexo 9.

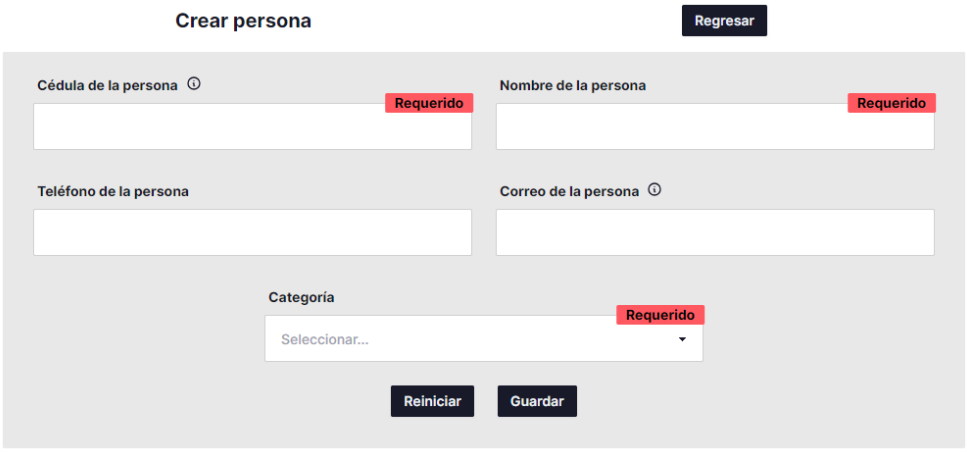


Figura 44. Interfaz de usuario para F7.1 Crear persona.

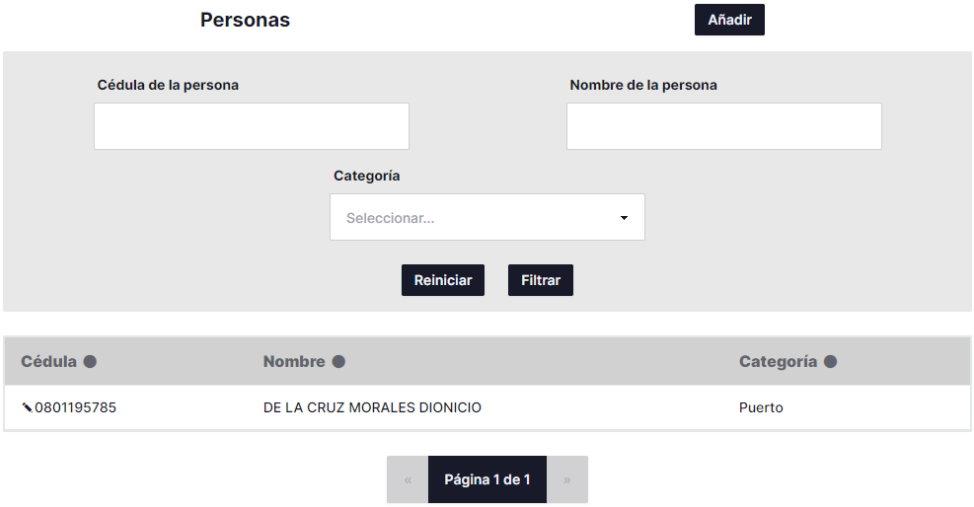


Figura 45. Interfaz de usuario para F7.2 Consultar persona.

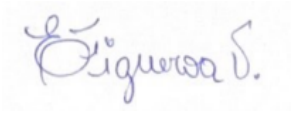
Figura 46. Interfaz de usuario para F7.3 Modificar persona.

Figura 47. Interfaz de usuario para F7.4 Eliminar persona.

Tabla 22

Pruebas de usuario para el requerimiento Gestión de personas

Caso de prueba	
Código de prueba	7
Historia de usuario	7. Gestión de personas
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Base de datos” de la barra de navegación superior. 3. Pulsar el botón “Personas” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar”

	7. En la misma página modificar la persona creada anteriormente. 8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado. 10. Buscar la persona modificada anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 11. En la página de la persona seleccionado hacer clic en el botón “Eliminar” del encabezado. 12. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar una persona sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar personas siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de personas. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de personas funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de personas.

Fuente: Producción propia

5.6.2. F8 Gestión de embarcaciones

Permite al usuario gestionar todos los aspectos de las embarcaciones, incluyendo su registro y la información asociada a propietarios y tripulantes. El código para esta funcionalidad se encuentra en el Anexo 10.

Figura 48. Interfaz de usuario para F8.1 Crear embarcación.

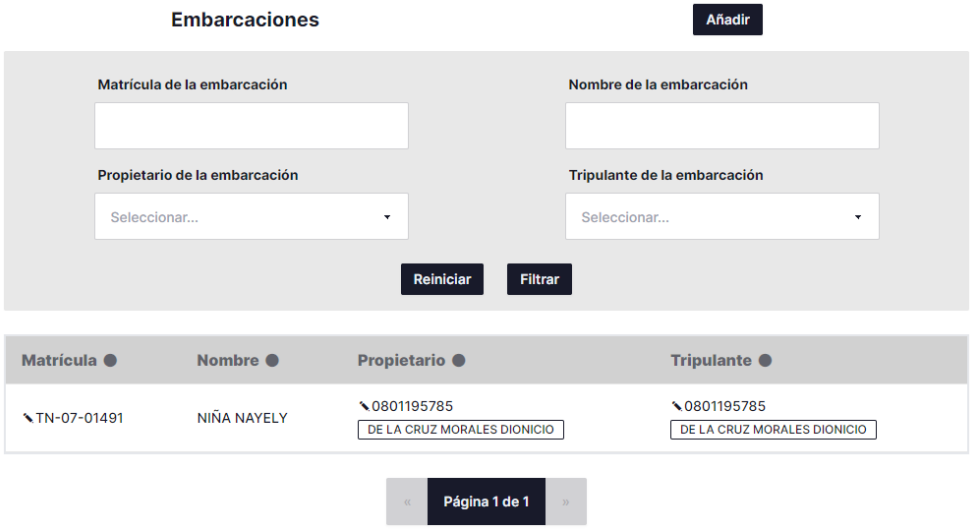


Figura 49. Interfaz de usuario para F8.2 Consultar embarcación.

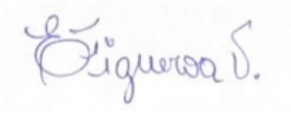
Figura 50. Interfaz de usuario para F8.3 Modificar embarcación.

Figura 51. Interfaz de usuario para F8.4 Eliminar embarcación.

Tabla 23

Pruebas de usuario para el requerimiento Gestión de embarcaciones

	Caso de prueba
Código de prueba	8
Historia de usuario	8. Gestión de embarcaciones
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Base de datos” de la barra de navegación superior. 3. Pulsar el botón “Embarcaciones” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar” 7. En la misma página modificar la embarcación creada anteriormente. 8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado. 10. Buscar la embarcación modificada anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 11. En la página de la embarcación seleccionado hacer clic en el botón “Eliminar” del encabezado. 12. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar una embarcación sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar embarcaciones siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado	El sistema funcionó según lo esperado, permitiendo la creación,

obtenido	modificación y eliminación de embarcaciones. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de motores funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de motores.

Fuente: Producción propia

5.6.3. F9 Gestión de motores

Esta funcionalidad permite al usuario gestionar los motores asignados a las embarcaciones correspondientes. El código para esta funcionalidad se encuentra en el Anexo 11.

Figura 52. Interfaz de usuario para F9.1 Crear motor.

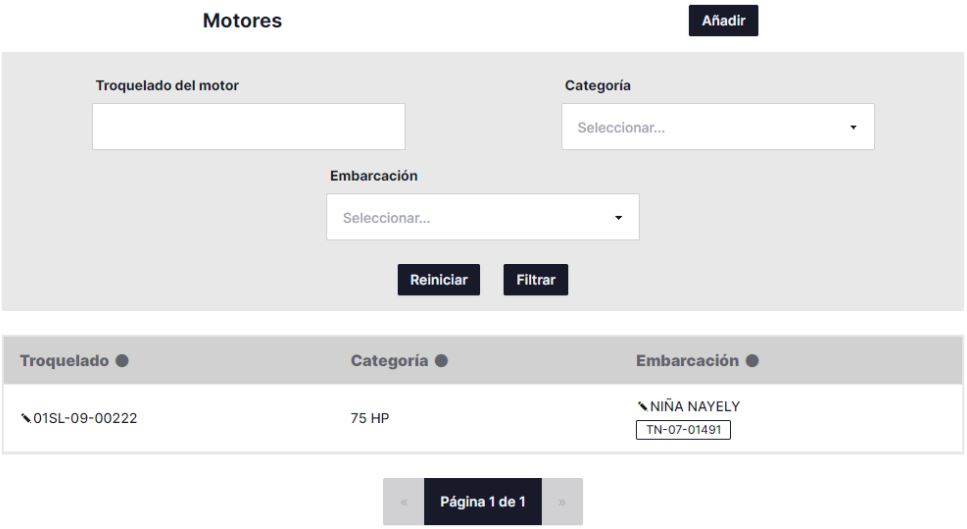


Figura 53. Interfaz de usuario para F9.2 Consultar motor.

Figura 54. Interfaz de usuario para F9.3 Modificar motor.

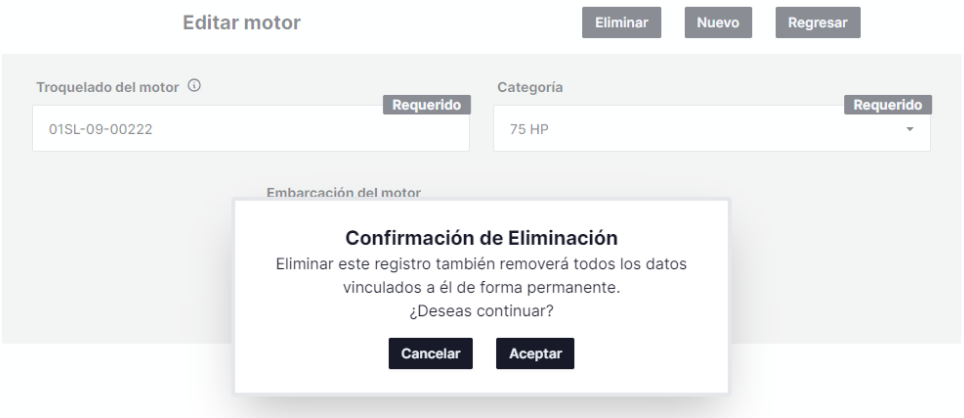
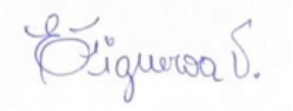


Figura 55. Interfaz de usuario para F9.4 Eliminar motor.

Tabla 24

Pruebas de usuario para el requerimiento Gestión de motores

Caso de prueba	
Código de prueba	9
Historia de usuario	9. Gestión de motores
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Base de datos” de la barra de navegación superior. 3. Pulsar el botón “Motores” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar” 7. En la misma página modificar el motor creado anteriormente. 8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado. 10. Buscar el motor modificado anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 11. En la página del motor seleccionado hacer clic en el botón “Eliminar” del encabezado. 12. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar un motor sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar motores siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de motores. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de	La prueba fue exitosa. Todas las funcionalidades de gestión de zarpes

resultados	funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de zarpes.

Fuente: Producción propia

5.6.4. F10 Gestión de zarpes

Facilita la creación y gestión de registros de zarpes, asegurando que todas las autorizaciones se documenten correctamente. El código para esta funcionalidad se encuentra en el Anexo 12.

Figura 56. Interfaz de usuario para F10.1 Crear zarpe.

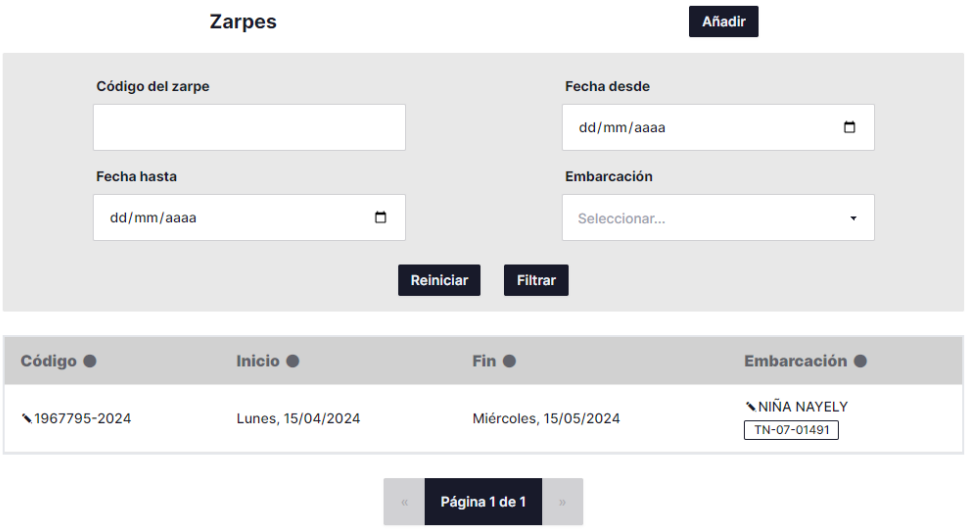


Figura 57. Interfaz de usuario para F10.2 Consultar zarpe.

Figura 58. Interfaz de usuario para F10.3 Modificar zarpe.

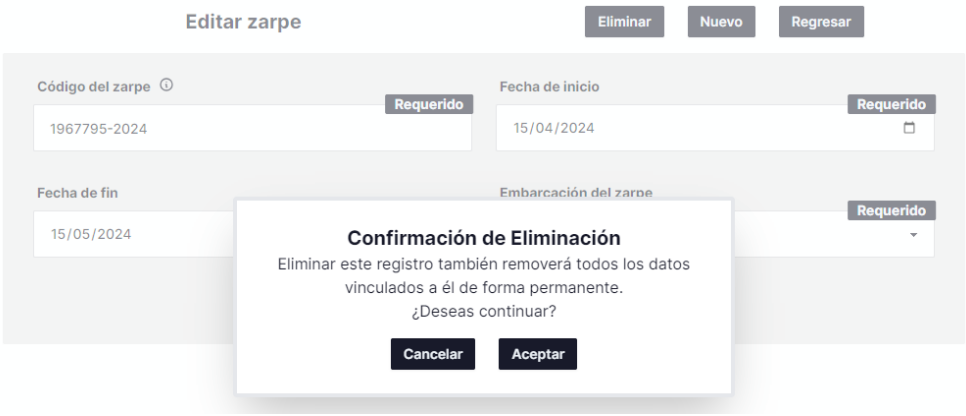
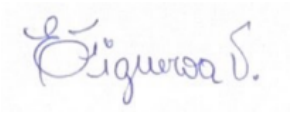


Figura 59. Interfaz de usuario para F10.4 Eliminar zarpe.

Tabla 25

Pruebas de usuario para el requerimiento Gestión de zarpes

	Caso de prueba
Código de prueba	10
Historia de usuario	10. Gestión de zarpes
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Base de datos” de la barra de navegación superior. 3. Pulsar el botón “Zarpes” 4. Dar clic en el botón “Añadir” del encabezado de la página. 5. Completar la información de los campos requeridos. 6. Dar clic en el botón “Guardar” 7. En la misma página modificar el zarpe creado anteriormente. 8. Dar clic en el botón “Guardar” 9. En la misma página dar clic en el botón “Regresar” del encabezado.

	10. Buscar el zarpe modificado anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 11. En la página del zarpe seleccionado hacer clic en el botón “Eliminar” del encabezado. 12. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar un zarpe sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede visualizar, modificar y eliminar zarpes siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de zarpes. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de zarpes funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de zarpes.

Fuente: Producción propia

5.7. ITERACIÓN 4

En esta iteración final, se implementaron las funcionalidades críticas para la gestión de ventas y reportes, permitiendo la gestión de transacciones y la generación de reportes para el análisis y la toma de decisiones.

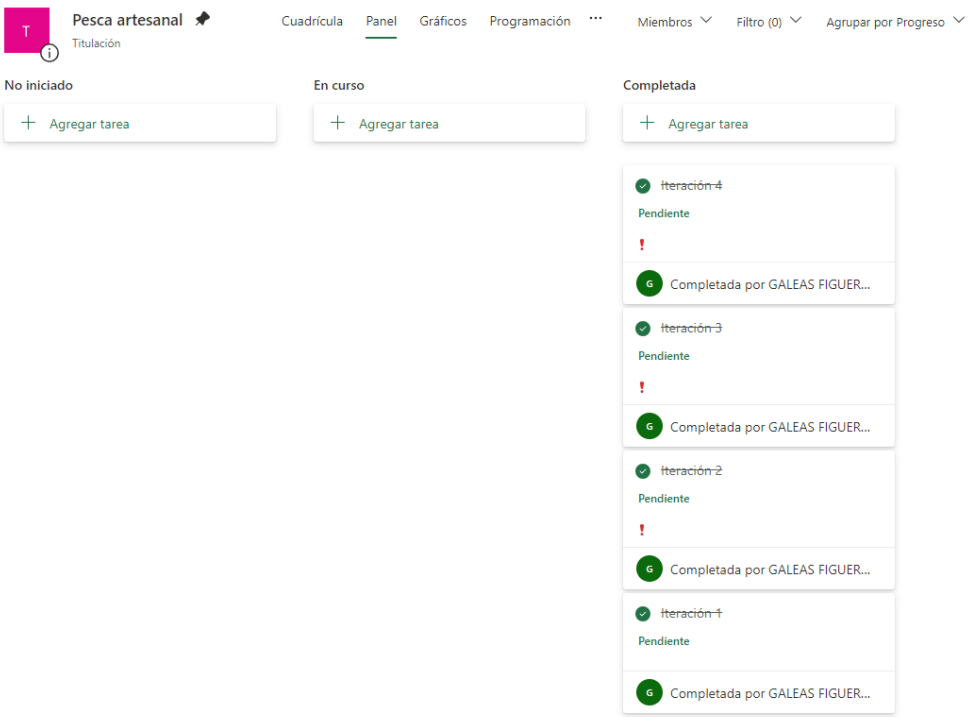


Figura 60. Cuarta actualización del cronograma basado en el progreso de las iteraciones.

5.7.1. F11 Gestión de ventas

Esta funcionalidad permite al usuario gestionar todos los aspectos de las ventas, incluyendo el registro y seguimiento de transacciones. El código para esta funcionalidad se encuentra en el Anexo 13.

Figura 61. Interfaz de usuario para F11.1 Crear venta.

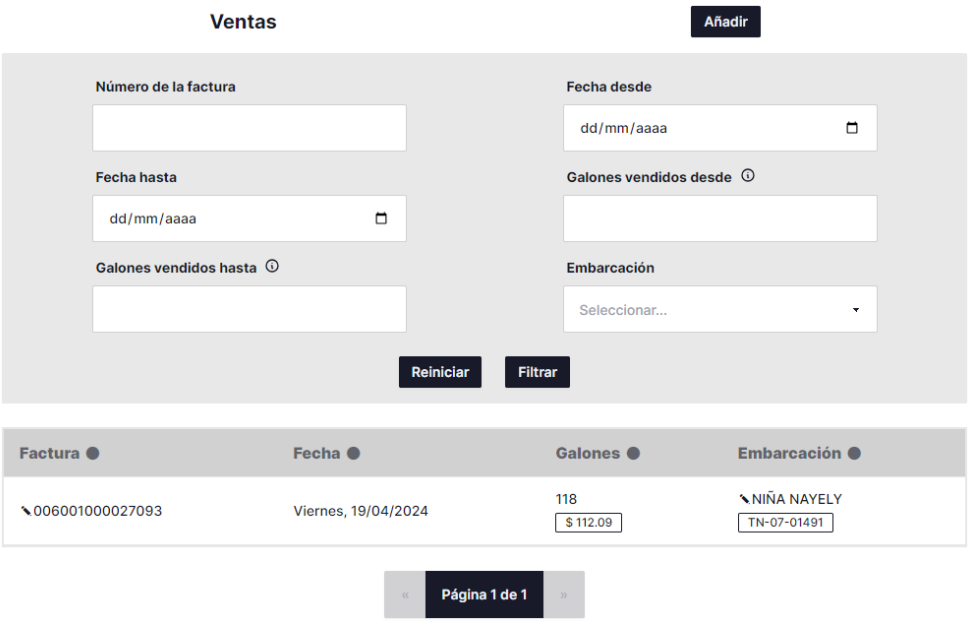


Figura 62. Interfaz de usuario para F11.2 Consultar venta.

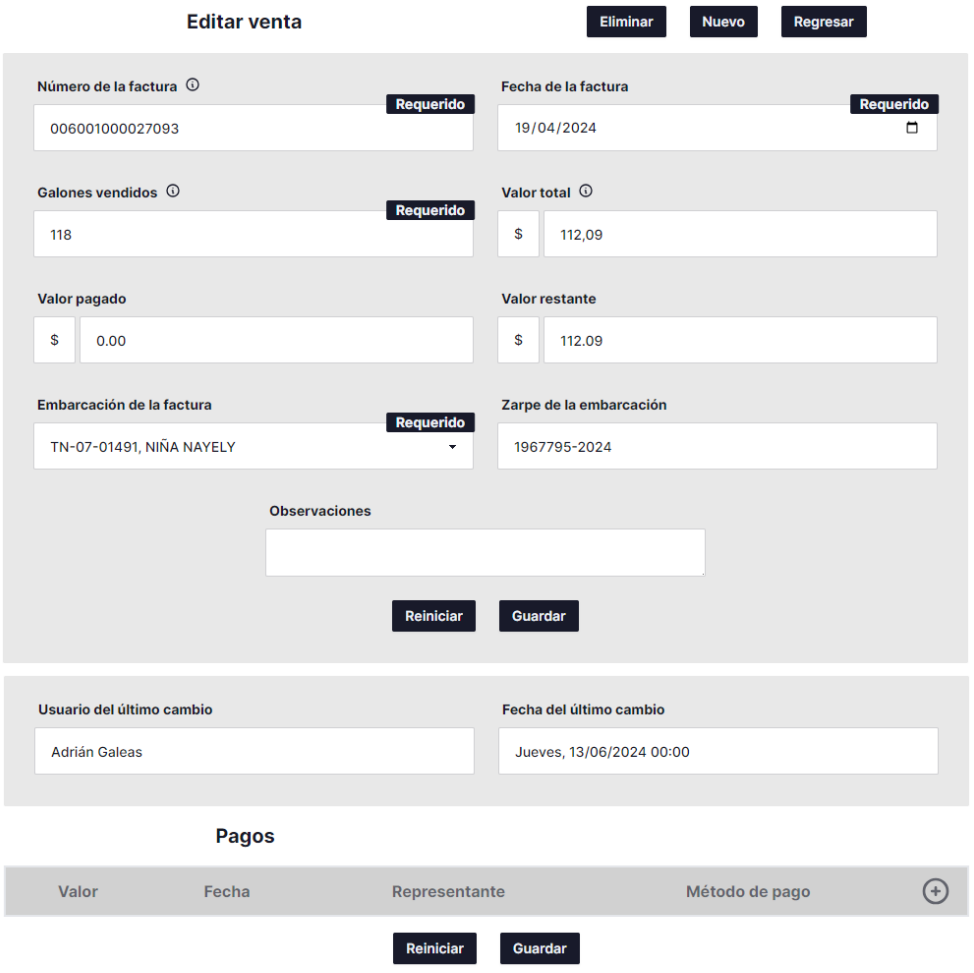


Figura 63. Interfaz de usuario para F11.3 Modificar venta.

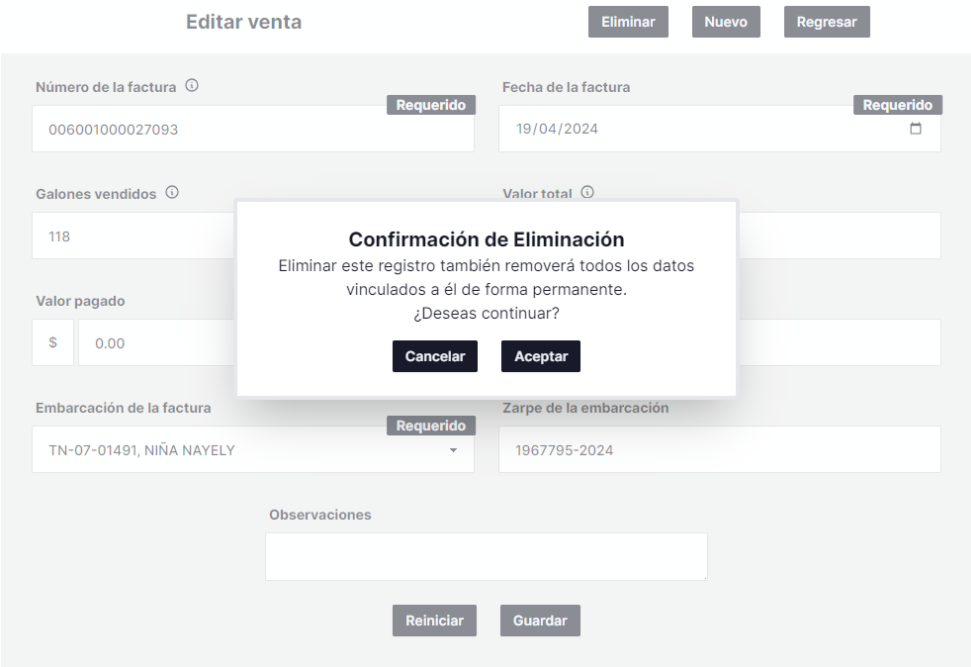
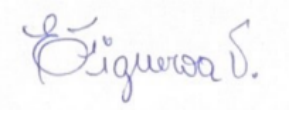


Figura 64. Interfaz de usuario para F11.4 Eliminar venta.

Tabla 26

Pruebas de usuario para el requerimiento Gestión de ventas

Caso de prueba	
Código de prueba	11
Historia de usuario	11. Gestión de ventas
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Ventas” de la barra de navegación superior. 3. Dar clic en el botón “Añadir” del encabezado de la página. 4. Completar la información de los campos requeridos. 5. Dar clic en el botón “Guardar” 6. En la misma página modificar la venta creada anteriormente. 7. Dar clic en el botón “Guardar” 8. En la misma página dar clic en el botón “Regresar” del encabezado. 9. Buscar la venta modificada anteriormente en la tabla inferior y seleccionarlo dando clic en el primer campo de la fila. 10. En la página de la venta seleccionada hacer clic en el botón “Eliminar” del encabezado. 11. Leer la cláusula y dar clic en el botón “Aceptar”.
Resultado esperado	El sistema permite agregar, modificar y eliminar una venta sin errores, y los cambios reflejan correctamente en la base de datos. El actor puede

	visualizar, modificar y eliminar ventas siguiendo el flujo de acciones sin interrupciones o errores en el proceso.
Resultado obtenido	El sistema funcionó según lo esperado, permitiendo la creación, modificación y eliminación de ventas. Todos los cambios fueron reflejados adecuadamente en la interfaz y confirmados en la base de datos.
Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. Todas las funcionalidades de gestión de ventas funcionaron como se esperaba, demostrando la integridad y eficacia del sistema en la manipulación de datos de ventas.

Fuente: Producción propia

5.7.2. F12 Gestión de reportes

Permite al usuario generar y personalizar reportes, facilitando el acceso a información clave para la toma de decisiones. El código para esta funcionalidad se encuentra en el Anexo 14.

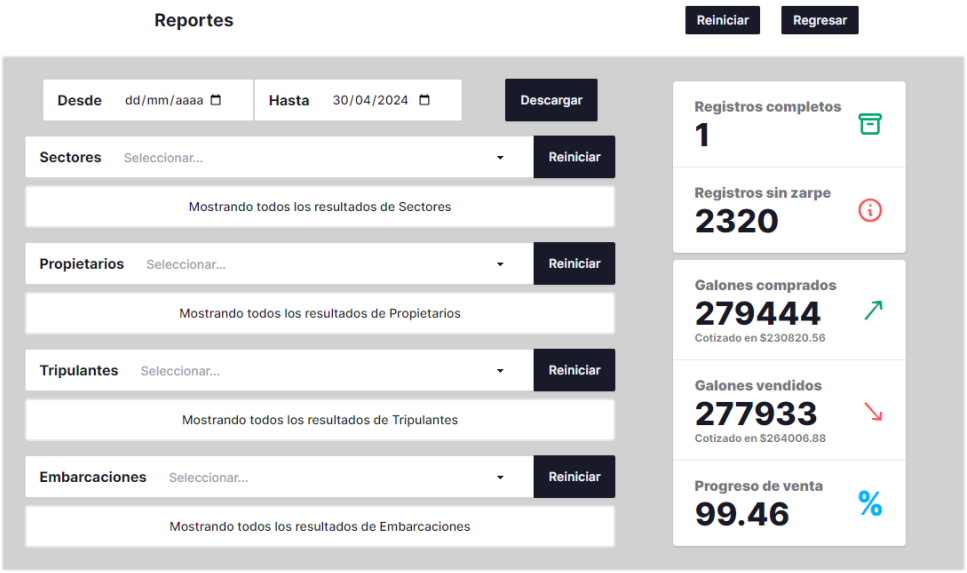


Figura 65. Interfaz de usuario para F12.1 Visualizar reportes por filtros.

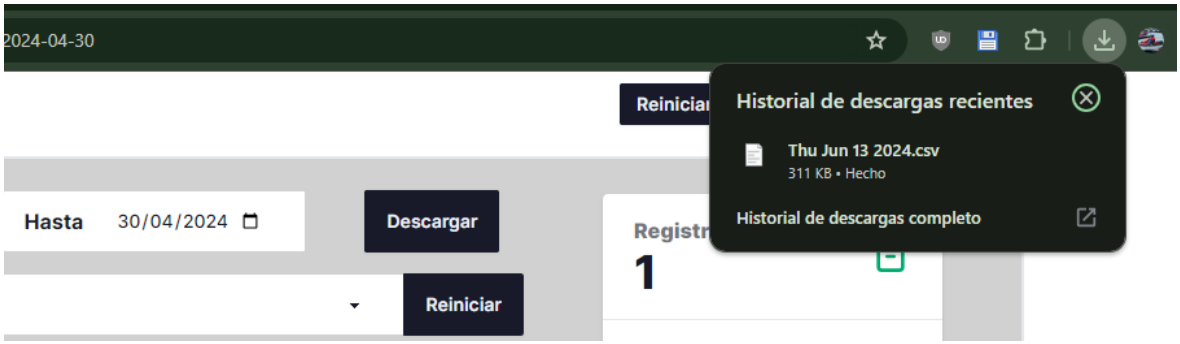
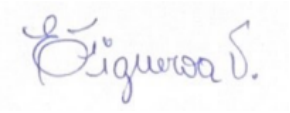


Figura 66. Interfaz de usuario para F12.2 Descargar reporte.

Tabla 27

Pruebas de usuario para el requerimiento Gestión de reportes

	Caso de prueba
Código de prueba	12
Actores	Usuario
Historia de usuario	12. Gestión de reportes
Pasos de Ejecución	1. Dirigirse a la página de inicio en el sistema. 2. Seleccionar la opción de “Reportes” de la barra de navegación superior. 3. Dar clic en el icono de calendario del campo “Desde” y seleccionar la fecha deseada. 4. Dar clic en el icono de calendario del campo “Hasta” y seleccionar la fecha deseada. 4. Con la barra lateral, desplazarse a la parte inferior de la página. 5. Visualizar los registros que coinciden con el criterio de búsqueda “Desde” y “Hasta”. 6. Con la barra lateral, desplazarse a la parte superior de la página. 7. Dar clic en el botón “Descargar” junto al criterio de búsqueda “Hasta” 8. Abrir el archivo descargado y confirmar los datos exportados.
Resultado esperado	Visualización del reporte general de operaciones en base a filtros y posterior descarga para procesos externos, asegurando que los datos sean correctos y completos.
Resultado obtenido	El sistema permitió visualizar y descargar los reportes de operaciones conforme a los filtros aplicados, y los datos exportados fueron precisos y completos.

Firma del desarrollador responsable	 Adrián Galeas Figueroa
Firma del evaluador de la prueba (cliente)	 Elsy Figueroa Villegas
Valoración de resultados	La prueba fue exitosa. La funcionalidad de gestión de reportes funcionó como se esperaba, demostrando que el sistema puede manejar eficientemente la generación y descarga de reportes basados en criterios específicos de fecha, proporcionando una herramienta efectiva y precisa para el análisis y la toma de decisiones.

Fuente: Producción propia

5.8. ENLACE PARA ENTRAR A LA APLICACIÓN

http://34.16.26.161

CAPÍTULO VI: CONCLUSIONES Y RECOMENDACIONES

6.1. CONCLUSIONES

1. El desarrollo del prototipo del sistema de información ha demostrado la importancia de las herramientas tecnológicas en asegurar el cumplimiento de las normativas gubernamentales, optimizando el control y la gestión en el sector de la pesca artesanal.
2. La adaptabilidad y la iteración rápida de la metodología Extreme Programming (XP) utilizada en el desarrollo del proyecto han demostrado ser cruciales para responder eficazmente a los cambios y requerimientos durante el ciclo de desarrollo del software.
3. Las historias de usuario detalladas reflejan un enfoque meticuloso para garantizar un acceso controlado y personalizado a las funcionalidades del sistema, enfatizando la importancia de la seguridad y la efectividad operativa.
4. La planificación y ejecución de iteraciones delinean un método sistemático para manejar la complejidad y los riesgos en el desarrollo del software, asegurando que cada fase contribuya eficazmente a la construcción de un sistema robusto y funcional.
5. El proceso de desarrollo y pruebas del sistema, ejecutado en iteraciones, ha permitido realizar ajustes basados en retroalimentación real, asegurando que el software final cumpla con las necesidades especificadas y sea robusto en su funcionamiento.
6. La funcionalidad de gestión de parámetros demuestra la importancia de permitir la configuración flexible del sistema, facilitando ajustes rápidos a los cambios del entorno sin intervención técnica compleja y sin entrar al código.

6.2. RECOMENDACIONES

1. Se recomienda aplicar una metodología de desarrollo que permite planificar y organizar las diferentes etapas del ciclo de desarrollo de software, es así que para el desarrollo del presente trabajo de integración curricular se utilizó la metodología XP que facilita la comunicación constante, la retroalimentación rápida, y el desarrollo iterativo.

2. Es vital generar todos los entregables o productos contemplados en la metodología seleccionada, sin descuidar la documentación que es esencial para el desarrollo y mantenimiento del software.
3. Seleccionar herramientas de diseño y desarrollo de software para crear una interfaz de usuario intuitiva para el manejo de la aplicación, emplear herramientas de desarrollo colaborativo y control de versiones, como Git y GitHub, para mejorar la gestión de las versiones del software y coordinar eficazmente el proyecto, incluso en entornos de desarrollo individual.
4. La gestión de parámetros externos permite a los usuarios finales realizar cambios de manera independiente y segura, aumentando la eficacia operativa y reduciendo la necesidad de soporte técnico externo.
5. Realizar pruebas de la funcionalidad de la aplicación, juntamente con el usuario y en forma progresiva, ayudan a crear aplicaciones eficientes que cumplen los requerimientos funcionales.

CAPÍTULO VII: BIBLIOGRAFÍA

Acharya, B., & Kumar Sahu, P. (2020). Software development life cycle models: A review paper. *International Journal of Advanced Research in Engineering and Technology*, 12. Recuperado el 21 de Abril de 2024

Agile Alliance. (18 de Octubre de 2023). *What is Extreme Programming (XP)?* Recuperado el 12 de Abril de 2024, de Agile Alliance: https://www.agilealliance.org/glossary/xp/

Al-Saqqa, S. (2020). Agile software development: Methodologies and trends. *International Journal of Interactive Mobile Technologies*, 14. doi:https://doi.org/10.3991/ijim.v14i11.13269

ARCERNNR. (31 de Agosto de 2023). *COMUNICADO OFICIAL*. Obtenido de La ARCERNNR ejecuta controles a la compra y transporte de combustible: https://controlrecursosyenergia.gob.ec/comunicado-oficial-la-arcernnr-ejecuta-controles-a-la-compra-y-transporte-de-combustible-modalidad-cuantia-domestica/

ARCERNNR. (s.f.). *Misión*. Recuperado el 8 de Septiembre de 2023, de Agencia de Regulación y Control de Energía y Recursos Naturales No Renovables: https://www.controlrecursosyenergia.gob.ec/mision/

Banerjee, S. (7 de Marzo de 2024). *Understanding the Life Cycle Phases of the Extreme Programming Development Model*. Recuperado el 25 de Abril de 2024, de RS Web Solutions: https://www.rswebsols.com/extreme-programming-development-model/

GoodFirms. (2024). *What is a Software Tools?* Recuperado el 12 de Abril de 2024, de GoodFirms: https://www.goodfirms.co/glossary/software-tools

INEC. (s.f.). *Agencia de Regulación y Control de Energía y Recursos Naturales no Renovables*. Recuperado el 17 de Noviembre de 2023, de Acerca de: https://anda.inec.gob.ec/anda/index.php/catalog/ARCERNNR/about

Jama Software. (7 de Abril de 2023). *Functional vs. Non-Functional Requirements*. Recuperado el 15 de Mayo de 2024, de Jama Software: https://www.jamasoftware.com/requirements-management-guide/writing-requirements/functional-vs-non-functional-requirements

Jeffries, R. (21 de Noviembre de 2001). *Essential XP: Documentation*. Recuperado el 25 de Abril de 2024, de RonJeffries: https://ronjeffries.com/xprog/articles/expdocumentationinxp/

Nagler, R. (2017). *Extreme Programming in Perl*. Recuperado el 6 de Mayo de 2024

PlantUML. (2009). *Cuál es el objetivo de PlantUML?* Recuperado el 24 de Mayo de 2024, de Frequently Asked Questions (FAQ): https://plantuml.com/es/faq

Simmons, L. (9 de Noviembre de 2023). *Front-End vs. Back-End: What's the Difference?* Recuperado el 14 de Diciembre de 2023, de ComputerScience: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/

Stsepanets, A. (31 de Enero de 2023). *Características de Microsoft Planner, sus ventajas y desventajas*. Recuperado el 23 de Abril de 2024, de GanttPRO: https://blog.ganttpro.com/es/caracteristicas-de-microsoft-planner-ventajas-y-desventajas/

Taylor, D. (30 de Diciembre de 2023). *What is Data Modelling? Types (Conceptual, Logical, Physical)*. Obtenido de Guru99: https://www.guru99.com/data-modelling-conceptual-logical.html

The PostgreSQL Global Development Group. (2024). *What is PostgreSQL?* Recuperado el 12 de Abril de 2024, de PostgreSQL About: https://www.postgresql.org/about/

Vercel, Inc. (2024). *About React and Next.js*. Recuperado el 12 de Abril de 2022, de Next.js: https://nextjs.org/learn/react-foundations/what-is-react-and-nextjs

Volle, A. (6 de Octubre de 2022). *Web application*. Recuperado el 18 de Noviembre de 2023, de Encyclopedia Britannica: https://www.britannica.com/topic/Web-application

Zwass, V. (10 de Noviembre de 2023). *information system*. Recuperado el 18 de Noviembre de 2023, de Science & Tech: https://www.britannica.com/topic/information-system

ANEXOS

Anexo 1, Glosario de términos.

1. **Autenticación**: Proceso de verificar la identidad de un usuario antes de permitirle acceso a un sistema.
2. **Capitanía**: Autoridad encargada de la administración y regulación de actividades marítimas en puertos y zonas costeras.
3. **Codificación**: Conversión de datos o información en un formato específico para procesamiento o transmisión.
4. **Desnormalización**: Proceso de optimización en bases de datos que reduce la cantidad de tablas y acelera las consultas al permitir redundancias controladas.
5. **Entidad**: Objeto o concepto del mundo real que posee datos y puede ser claramente identificado.
6. **Interfaz**: Medios por los cuales los usuarios interactúan con un sistema o dispositivo.
7. **Iteración**: Ciclo de desarrollo en metodologías ágiles que incluye planificación, ejecución y revisión.
8. **Normalización**: Proceso de organización de datos en una base de datos para reducir la redundancia y mejorar la integridad.
9. **Permisos**: Autorizaciones concedidas a usuarios para realizar acciones específicas dentro de un sistema.
10. **Prototipado**: Creación de una versión inicial de un sistema para probar y refinar conceptos.
11. **Regulación**: Conjunto de normas establecidas para controlar una actividad, en este contexto, la distribución y uso de combustible.
12. **Relación**: Conexión o asociación entre dos entidades en una base de datos.
13. **SGBD**: Acrónimo de Sistema de Gestión de Bases de Datos, software que facilita la creación y administración de bases de datos.
14. **Token**: Datos que sirven como credenciales de seguridad en comunicaciones electrónicas o sesiones de usuario.
15. **UML**: Acrónimo de Unified Modeling Language, lenguaje estándar para especificar, visualizar, construir y documentar los artefactos de un sistema de software.

Anexo 2, Circular de la ARCERNNR sobre requisitos para el despacho de combustible.

Gobierno del Ecuador
GUILLERMO LASSO
PRESIDENTE

CIRCULAR

Se comunica que conforme al Art. 9 del Acuerdo Interinstitucional "INSTRUCTIVO PARA LA DISTRIBUCIÓN DE COMBUSTIBLE A LAS EMBARCACIONES QUE UTILIZAN GASOLINA DE PESCA ARTESANAL PARA SU ACTIVIDAD" indica lo siguiente:

Documentos para el despacho de combustible para las embarcaciones que utilizan GPA a través de los centros de distribución:

El centro de distribución procederá al **despacho de GPA** a las embarcaciones, siempre que se hayan presentado los siguientes documentos:

- Cédula de ciudadanía del armador de la embarcación o su delegado;
- Autorización formal del armador, en caso de ser un delegado que realice el retiro; y,
- Documento de zarpe de la embarcación.

Esta dirección distrital hace un recordatorio de las directrices a fin de que sea de cumplimiento.

Agencia de Regulación y Control de Energía y Recursos Naturales No Renovables
Dirección: Avenida Naciones Unidas E7-71 y Av. De Los Shyris
Código postal: 170506 / Quito-Ecuador
Teléfono: +593-2 226 8744
www.controlrecursosyenergia.gob.ec

República del Ecuador

Anexo 3, Código del requerimiento funcional F1 Gestión de sesiones.

```
'use client'
import { useContext, useState } from 'react'
import { useRouter } from 'next/navigation'
import { logIn } from '@/lib/user'
import { EnvironmentContext } from '@/components/environment/Environment'

export default function Page() {
    const { addAlert } = useContext(EnvironmentContext)
    const { replace } = useRouter()

    const [form, setForm] = useState({ id: '', pass: '' })
    const [showPass, setShowPass] = useState(false)
    const [loading, setLoading] = useState(false)
    const [theme, setTheme] = useState((hora => hora >= 18 || hora < 6)(new Date().getHours()))

    const handleForm = (event) => {
        setForm({
            ...form,
            [event.target.name]: event.target.value
        })
    }

    const handleTheme = () => {
        setTheme(!theme)
    }

    return (
        <div className='flex flex-grow flex-col bg-base-100 w-auto justify-center items-center'>
            <div className='card w-full max-w-md bg-base-200 shadow-xl'>
                <div className='card-body w-auto'>
                    <h2 className='card-title text-2xl justify-center'>Gasolinera del Puerto</h2>
                    <label className='form-control w-full'>
                        <div className='label'>
                            <span className='label-text font-bold text-lg'>Usuario</span>
                        </div>
                        <label className='input input-bordered flex items-center gap-2 font-medium'>
                            <input
                                type='text'
                                name='id'
                                maxLength={32}
                                onChange={handleForm}
                                placeholder='Usuario'
                                className='grow'
                            />
                        </label>
                    </label>
                    <label className='form-control w-full'>
                        <div className='label'>
                            <span className='label-text font-bold text-lg'>Contraseña</span>
                        </div>
                        <label className='input input-bordered flex items-center gap-2'>
                            <input
                                name='pass'
                                type={showPass ? 'text' : 'password'}
                                maxLength={24}
                                onChange={handleForm}
                                placeholder='Contraseña'
                                className='grow'
                            />
                            <label className='badge badge-neutral swap'>
                                <input
                                    type='checkbox'
                                    value={showPass}
                                    onChange={() => setShowPass(!showPass)}
                                />
                                <div className='swap-on font-bold'>Ocultar</div>
                                <div className='swap-off font-bold'>Mostrar</div>
                            </label>
                        </label>
                    </label>
                    <div className='card-actions justify-between items-center mt-6'>
                        <label className='swap swap-rotate'>
                            <input
                                type='checkbox'
                                className='theme-controller'
                                value='night'
                                checked={theme}
                                onChange={handleTheme}
                            />
                        </label>
                        <button
                            onClick={async () => {
                                setLoading(true)
                                const error = await logIn(form)
                                error ? addAlert({ error: error }) : replace('/')
                                setLoading(false)
                            }}
                            disabled={loading}
                            className='btn btn-neutral text-lg'
                        >
                            Iniciar Sesión
                        </button>
                    </div>
                </div>
            </div>
        </div>
    )
}

```

```
'use client'
import { useContext } from 'react'
import { useRouter } from 'next/navigation'
import { logOut } from '@/lib/user'
import { EnvironmentContext } from '@/components/environment/Environment'

export default function NavLogOut() {
    const { addAlert } = useContext(EnvironmentContext)
    const { replace } = useRouter()
    return (
        <button
            onClick={async () => {
                const error = await logOut()
                error ? addAlert({ error: error }) : replace('/')
            }}
            className='text-lg justify-center'
        >
            Cerrar sesión
        </button>
    )
}

```

Anexo 4, Código del requerimiento funcional F2 Gestión de roles de usuario.

```
export async function Roles({ searchParams, authorizations: { Create } }) {
    const roles = await getRoles(searchParams)

    return (
        <Container page>
            <Search>
                <Search.Header
                    title='Roles de usuario'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='nombre'
                        label='Nombre del rol'
                    />
                    <Search.Box
                        label='Perfil de Administrador'
                    >
                        <Search.Radio
                            property='administrador'
                            value='true'
                            label='Activo'
                        />
                        <Search.Radio
                            property='administrador'
                            value='false'
                            label='Inactivo'
                        />
                    </Search.Box>
                    <Search.Box
                        label='Perfil de Usuario'
                    >
                        <Search.Radio
                            property='usuario'
                            value='true'
                            label='Activo'
                        />
                        <Search.Radio
                            property='usuario'
                            value='false'
                            label='Inactivo'
                        />
                    </Search.Box>
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortNombre'
                            label='Rol'
                        />
                        <Table.Column
                            sort='sortAdministrador'
                            label='Perfil Administrador'
                        />
                        <Table.Column
                            sort='sortUsuario'
                            label='Perfil Usuario'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={roles}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={roles.length}
                />
            </Table>
        </Container>
    )
}
```

```jsx
export async function Rol({ searchParams, resource, authorizations: { Create, Update, Delete } }) {
    const rol = resource === '~'
        ? null
        : await readRol(resource)

    if (rol || resource === '~') {
        const perfiles = [
            { value: 'administrador', label: 'Administrador' },
            { value: 'usuario', label: 'Usuario' }
        ]
        const perfil = perfiles.find(option => option.value === searchParams.perfil)

        const autorizaciones = rol && perfil
            ? await readAutorizaciones(rol.id, perfil.value)
            : []

        const modulos = rol && perfil
            ? await getArrayModulos(perfil.value)
            : []

        return (
            <Container
            type='page'
        >
                <Form
                    Data={rol}
                    Create={Create ? createRol : null}
                    Update={Update ? updateRol : null}
                    Delete={Delete ? deleteRol : null}
                    pathname='/administrador/roles'
                >
                    <Form.Header
                        title={`${!rol ? 'Crear' : Update ? 'Editar' : 'Visualizar'} rol de usuario`}
                    />
                    <Form.Body>
                        <Form.String
                            property='nombre'
                            required
                            label='Nombre del rol'
                        />
                        <Form.Box
                            label='Perfil de Administrador'
                        >
                            <Form.Radio
                                property='administrador'
                                value='true'
                                label='Activo'
                            />
                            <Form.Radio
                                property='administrador'
                                value='false'
                                label='Inactivo'
                            />
                        </Form.Box>
                        <Form.Box
                            label='Perfil de Usuario'
                        >
                            <Form.Radio
                                property='usuario'
                                value='true'
                                label='Activo'
                            />
                            <Form.Radio
                                property='usuario'
                                value='false'
                                label='Inactivo'
                            />
                        </Form.Box>
                        {
                            rol && (
                                <Form.Field
                                    property='date'
                                    label='Fecha de ultima modificación'
                                />
                            )
                        }
                    </Form.Body>
                </Form>
```

```
                {
                    rol && (
                        <Multiple
                            parent={{ rol: rol.id, perfil: perfil?.value }}
                            childs={autorizaciones}
                            action={Update && perfil ? updateAutorizaciones : null}
                        >
                            <Multiple.Header
                                title={perfil?.label || 'Permisos'}
                                button='Seleccionar Perfil'
                                param='perfil'
                                options={perfiles}
                            />
                            <Multiple.Body
                                columns={[
                                    'Modulos',
                                    'Crear',
                                    'Visualizar',
                                    'Editar',
                                    'Eliminar'
                                ]}
                            >
                                <Multiple.Autocomplete
                                    property='ruta'
                                    options={modulos}
                                    required
                                />
                                <Multiple.CheckBox
                                    property='Create'
                                />
                                <Multiple.CheckBox
                                    property='Read'
                                />
                                <Multiple.CheckBox
                                    property='Update'
                                />
                                <Multiple.CheckBox
                                    property='Delete'
                                />
                            </Multiple.Body>
                        </Multiple>
                    )
                }
            </Container>
        )
    } else {
        return <Info message={`No se encontró el rol de usuario con código \'${resource}\'`} />
    }
}
```

Anexo 5, Código del requerimiento funcional F3 Gestión de usuarios.

```
export async function Usuarios({ searchParams, authorizations: { Create } }) {
    const usuarios = await getUsuarios(searchParams)
    const roles = await getArrayRoles()

    return (
        <Container
            type='page'
        >
            <Search>
                <Search.Header
                    title='Usuarios'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='id'
                        label='Código de usuario'
                    />
                    <Search.String
                        property='nombre'
                        label='Nombre de usuario'
                    />
                    <Search.Autocomplete
                        property='rol'
                        options={roles}
                        label='Rol de usuario'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortID'
                            label='Usuario'
                        />
                        <Table.Column
                            sort='sortNombre'
                            label='Nombre'
                        />
                        <Table.Column
                            sort='sortRol'
                            label='Rol'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={usuarios}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={usuarios.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Usuario({ resource, authorizations: { Create, Update, Delete } }) {
    const usuario = resource === '~'
        ? null
        : await readUsuario(resource)

    if (usuario || resource === '~') {
        const roles = await getArrayRoles()

        return (
            <Container
            type='page'
        >
                <Form
                    Data={usuario}
                    Create={Create ? createUsuario : null}
                    Update={Update ? updateUsuario : null}
                    Delete={Delete ? deleteUsuario : null}
                    pathname='/administrador/usuarios'
                    redirect='id'
                >
                    <Form.Header
                        title={`${!usuario ? 'Crear' : Update ? 'Editar' : 'Visualizar'} usuario`}
                    />
                    <Form.Body>
                        <Form.PrimaryKey
                            property='id'
                            label='Código de usuario'
                        />
                        <Form.DefaultPass
                            property='pass'
                        />
                        <Form.String
                            property='nombre'
                            required
                            label='Nombre de usuario'
                        />
                        <Form.Box
                            label='Estado'
                        >
                            <Form.Radio
                                property='activo'
                                value='true'
                                label='Activo'
                            />
                            <Form.Radio
                                property='activo'
                                value='false'
                                label='Inactivo'
                            />
                        </Form.Box>
                        <Form.Autocomplete
                            property='rol'
                            options={roles}
                            required
                            label='Rol de usuario'
                        />
                        {
                            usuario && (
                                <Form.Field
                                    property='date'
                                    label='Fecha de ultima modificación'
                                />
                            )
                        }
                    </Form.Body>
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró el usuario con código \'${resource}\'`} />
    }
}
```

Anexo 6, Código del requerimiento funcional F4 Gestión de categorías.

```
export async function Categorias({ searchParams, authorizations: { Create } }) {
    const categorias = await getCategorias(searchParams)

    return (
        <Container page>
            <Search>
                <Search.Header
                    title='Cateogrías'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='nombre'
                        label='Nombre de la categoría'
                    />
                    <Search.Options
                        property='modulo'
                        options={modulos}
                        label='Módulo de la categoría'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortNombre'
                            label='Nombre de la categoría'
                        />
                        <Table.Column
                            sort='sortModulo'
                            label='Módulo de la categoría'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={categorias}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={categorias.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Categoria({ resource, authorizations: { Create, Update, Delete } }) {
    const categoria = resource === '~'
        ? null
        : await readCategoria(resource)

    if (categoria || resource === '~') {
        return (
            <Container page>
                <Form
                    Data={categoria}
                    Create={Create ? createCategoria : null}
                    Update={Update ? updateCategoria : null}
                    Delete={Delete ? deleteCategoria : null}
                    pathname='/administrador/categorias'
                >
                    <Form.Header
                        title={`${!categoria ? 'Crear' : Update ? 'Editar' : 'Visualizar'} categoría`}
                    />
                    <Form.Body>
                        <Form.String
                            property='nombre'
                            required
                            label='Nombre de la categoría'
                        />
                        <Form.Options
                            property='modulo'
                            backup='modulo'
                            options={modulos}
                            required
                            label='Módulo de la categoría'
                        />
                    </Form.Body>
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró la categoría con código \'${resource}\'`} />
    }
}
```

Anexo 7, Código del requerimiento funcional F5 Gestión de parámetros.

```
export async function Parametros({ searchParams, authorizations: { Create } }) {
    const parametros = await getParametros(searchParams)

    return (
        <Container page>
            <Search>
                <Search.Header
                    title='Parámetros'
                    Create={Create}
                />
                <Search.Body>
                    <Search.Number
                        property='minPrecio'
                        type='smFloat'
                        label='Precio mínimo de venta'
                    />
                    <Search.Number
                        property='maxPrecio'
                        type='smFloat'
                        label='Precio máximo de venta'
                    />
                    <Search.Number
                        property='iva'
                        type='mdFloat'
                        label='IVA'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortPrecio'
                            label='Precio de venta'
                        />
                        <Table.Column
                            sort='sortIVA'
                            label='IVA'
                        />
                        <Table.Column
                            sort='sortActiva'
                            label='Estado'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={parametros}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={parametros.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Parametro({ resource, authorizations: { Create, Update, Delete } }) {
    const parametro = resource === '~'
        ? null
        : await readParametro(resource)

    if (parametro || resource === '~') {
        return (
            <Container page>
                <Form
                    Data={parametro}
                    Create={Create ? createParametro : null}
                    Update={Update ? updateParametro : null}
                    Delete={Delete ? deleteParametro : null}
                    pathname='/administrador/parametros'
                >
                    <Form.Header
                        title={`${!parametro ? 'Crear' : Update ? 'Editar' : 'Visualizar'} parámetro`}
                    />
                    <Form.Body>
                        <Form.Number
                            property='precio'
                            type='smFloat'
                            required
                            label='Precio de venta del parámetro'
                            prefix='$'
                        />
                        <Form.Number
                            property='iva'
                            type='mdFloat'
                            required
                            label='IVA del parámetro'
                            prefix='%'
                        />
                        <Form.Box
                            label='Estado'
                        >
                            <Form.Radio
                                property='activa'
                                value='true'
                                label='Activa'
                            />
                            <Form.Radio
                                property='activa'
                                value='false'
                                label='Inactiva'
                            />
                        </Form.Box>
                    </Form.Body>
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró el parámetro con código \'${resource}\'`} />
    }
}
```

Anexo 8, Código del requerimiento funcional F6 Gestión de compras.

```
export async function Compras({ searchParams, authorizations: { Create } }) {
    const compras = await getCompras(searchParams)

    return (
        <Container
            type='page'
        >
            <Search>
                <Search.Header
                    title='Compras'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='id'
                        label='Número de la factura'
                    />
                    <Search.Date
                        property='inicio'
                        max='fin'
                        label='Fecha de la compra desde'
                    />
                    <Search.Date
                        property='fin'
                        min='inicio'
                        label='Fecha de la compra hasta'
                    />
                    <Search.Number
                        property='minGalones'
                        type='xlInt'
                        max='maxGalones'
                        label='Galones comprados desde'
                    />
                    <Search.Number
                        property='maxGalones'
                        type='xlInt'
                        min='minGalones'
                        label='Galones comprados hasta'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortID'
                            label='Factura'
                        />
                        <Table.Column
                            sort='sortFecha'
                            label='Fecha'
                        />
                        <Table.Column
                            sort='sortGalones'
                            label='Galones'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={compras}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={compras.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Compra({ resource, authorizations: { Create, Update, Delete } }) {
    const compra = resource === '~'
        ? null
        : await readCompra(resource)

    if (compra || resource === '~') {
        return (
            <Container
                type='page'
            >
                <Form
                    Data={compra}
                    Create={Create ? createCompra : null}
                    Update={Update ? updateCompra : null}
                    Delete={Delete ? deleteCompra : null}
                    pathname='/administrador/compras'
                    redirect='id'
                >
                    <Form.Header
                        title={`${!compra ? 'Crear' : Update ? 'Editar' : 'Visualizar'} compra`}
                    />
                    <Form.Body>
                        <Form.PrimaryKey
                            property='id'
                            label='Número de la factura'
                        />
                        <Form.Date
                            property='fecha'
                            required
                            label='Fecha de la compra'
                        />
                        <Form.Number
                            property='galones'
                            type='xlInt'
                            required
                            label='Galones comprados'
                        />
                        {
                            compra && (
                                <Form.Number
                                    property='total'
                                    type='xlFloat'
                                    label='Valor total'
                                />
                            )
                        }
                        <Form.String
                            property='observaciones'
                            area
                            label='Observaciones'
                        />
                    </Form.Body>
                    {
                        compra && (
                            <Form.Body
                                hideActions
                            >
                                <Form.Field
                                    property='usuario'
                                    label='Usuario del último cambio'
                                />
                                <Form.Field
                                    property='date'
                                    label='Fecha del último cambio'
                                />
                            </Form.Body>
                        )
                    }
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró la compra con código \'${resource}\'`} />
    }
}
```

Anexo 9, Código del requerimiento funcional F7 Gestión de personas.

```
export async function Personas({ searchParams, authorizations: { Create } }) {
    const personas = await getPersonas(searchParams)
    const categorias = await getArrayCategorias('personas')

    return (
        <Container
            type='page'
        >
            <Search>
                <Search.Header
                    title='Personas'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='id'
                        label='Cédula de la persona'
                    />
                    <Search.String
                        property='nombre'
                        label='Nombre de la persona'
                    />
                    <Search.Autocomplete
                        property='categoria'
                        options={categorias}
                        label='Categoría'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortID'
                            label='Cédula'
                        />
                        <Table.Column
                            sort='sortNombre'
                            label='Nombre'
                        />
                        <Table.Column
                            sort='sortCategoria'
                            label='Categoría'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={personas}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={personas.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Persona({ resource, authorizations: { Create, Update, Delete } }) {
    const persona = resource === '~'
        ? null
        : await readPersona(resource)

    if (persona || resource === '~') {
        const categorias = await getArrayCategorias('personas')

        return (
            <Container
            type='page'
        >
                <Form
                    Data={persona}
                    Create={Create ? createPersona : null}
                    Update={Update ? updatePersona : null}
                    Delete={Delete ? deletePersona : null}
                    pathname='/usuario/personas'
                    redirect='id'
                >
                    <Form.Header
                        title={`${!persona ? 'Crear' : Update ? 'Editar' : 'Visualizar'} persona`}
                    />
                    <Form.Body>
                        <Form.PrimaryKey
                            property='id'
                            label='Cédula de la persona'
                        />
                        <Form.String
                            property='nombre'
                            required
                            label='Nombre de la persona'
                        />
                        <Form.String
                            property='telefono'
                            type='tel'
                            label='Teléfono de la persona'
                        />
                        <Form.Email
                            property='correo'
                            label='Correo de la persona'
                        />
                        <Form.Autocomplete
                            property='categoria'
                            options={categorias}
                            required
                            label='Categoría'
                        />
                    </Form.Body>
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró la persona con cédula \'${resource}\'`} />
    }
}
```

Anexo 10, Código del requerimiento funcional F8 Gestión de embarcaciones.

```
export async function Embarcaciones({ searchParams, authorizations: { Create } }) {
    const embarcaciones = await getEmbarcaciones(searchParams)
    const personas = await getArrayPersonas()

    return (
        <Container
            type='page'
        >
            <Search>
                <Search.Header
                    title='Embarcaciones'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='id'
                        label='Matrícula de la embarcación'
                    />
                    <Search.String
                        property='nombre'
                        label='Nombre de la embarcación'
                    />
                    <Search.Autocomplete
                        property='propietario'
                        options={personas}
                        label='Propietario de la embarcación'
                    />
                    <Search.Autocomplete
                        property='tripulante'
                        options={personas}
                        label='Tripulante de la embarcación'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortID'
                            label='Matrícula'
                        />
                        <Table.Column
                            sort='sortNombre'
                            label='Nombre'
                        />
                        <Table.Column
                            sort='sortPropietario'
                            label='Propietario'
                        />
                        <Table.Column
                            sort='sortTripulante'
                            label='Tripulante'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={embarcaciones}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={embarcaciones.length}
                />
            </Table>
        </Container>
    )
}
```

```jsx
export async function Embarcacion({ resource, authorizations: { Create, Update, Delete } }) {
    const embarcacion = resource === '~'
        ? null
        : await readEmbarcacion(resource)

    if (embarcacion || resource === '~') {
        const personas = await getArrayPersonas()

        return (
            <Container
            type='page'
        >
                <Form
                    Data={embarcacion}
                    Create={Create ? createEmbarcacion : null}
                    Update={Update ? updateEmbarcacion : null}
                    Delete={Delete ? deleteEmbarcacion : null}
                    pathname='/usuario/embarcaciones'
                    redirect='id'
                >
                    <Form.Header
                        title={`${!embarcacion ? 'Crear' : Update ? 'Editar' : 'Visualizar'} embarcación`}
                    />
                    <Form.Body>
                        <Form.PrimaryKey
                            property='id'
                            required
                            label='Matrícula de la embarcación'
                        />
                        <Form.String
                            property='nombre'
                            required
                            label='Nombre de la embarcación'
                        />
                        <Form.Autocomplete
                            property='propietario'
                            options={personas}
                            required
                            label='Propietario de la embarcación'
                        />
                        <Form.Autocomplete
                            property='tripulante'
                            options={personas}
                            required
                            label='Tripulante de la embarcación'
                        />
                    </Form.Body>
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró la embarcación con matrícula \'${resource}\'`} />
    }
}
```

Anexo 11, Código del requerimiento funcional F9 Gestión de motores.

```
export async function Motores({ searchParams, authorizations: { Create } }) {
    const motores = await getMotores(searchParams)
    const categorias = await getArrayCategorias('motores')
    const embarcaciones = await getArrayEmbarcaciones()

    return (
        <Container
            type='page'
        >
            <Search>
                <Search.Header
                    title='Motores'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='id'
                        label='Troquelado del motor'
                    />
                    <Search.Autocomplete
                        property='categoria'
                        options={categorias}
                        label='Categoría'
                    />
                    <Search.Autocomplete
                        property='embarcacion'
                        options={embarcaciones}
                        label='Embarcación'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortID'
                            label='Troquelado'
                        />
                        <Table.Column
                            sort='sortCategoria'
                            label='Categoría'
                        />
                        <Table.Column
                            sort='sortEmbarcacion'
                            label='Embarcación'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={motores}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={motores.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Motor({ resource, authorizations: { Create, Update, Delete } }) {
    const motor = resource === '~'
        ? null
        : await readMotor(resource)

    if (motor || resource === '~') {
        const embarcaciones = await getArrayEmbarcaciones()
        const categorias = await getArrayCategorias('motores')

        return (
            <Container
            type='page'
        >
                <Form
                    Data={motor}
                    Create={Create ? createMotor : null}
                    Update={Update ? updateMotor : null}
                    Delete={Delete ? deleteMotor : null}
                    pathname='/usuario/motores'
                    redirect='id'
                >
                    <Form.Header
                        title={`${!motor ? 'Crear' : Update ? 'Editar' : 'Visualizar'} motor`}
                    />
                    <Form.Body>
                        <Form.PrimaryKey
                            property='id'
                            label='Troquelado del motor'
                        />
                        <Form.Autocomplete
                            property='categoria'
                            options={categorias}
                            required
                            label='Categoría'
                        />
                        <Form.Autocomplete
                            property='embarcacion'
                            options={embarcaciones}
                            required
                            label='Embarcación del motor'
                        />
                    </Form.Body>
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró el motor con troquelado \'${resource}\'`} />
    }
}
```

Anexo 12, Código del requerimiento funcional F10 Gestión de zarpes.

```
export async function Zarpes({ searchParams, authorizations: { Create } }) {
    const zarpes = await getZarpes(searchParams)
    const embarcaciones = await getArrayEmbarcaciones()

    return (
        <Container
            type='page'
        >
            <Search>
                <Search.Header
                    title='Zarpes'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='id'
                        label='Código del zarpe'
                    />
                    <Search.Date
                        property='inicio'
                        max='fin'
                        label='Fecha desde'
                    />
                    <Search.Date
                        property='fin'
                        min='inicio'
                        label='Fecha hasta'
                    />
                    <Search.Autocomplete
                        property='embarcacion'
                        options={embarcaciones}
                        label='Embarcación'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortID'
                            label='Código'
                        />
                        <Table.Column
                            sort='sortInicio'
                            label='Inicio'
                        />
                        <Table.Column
                            sort='sortFin'
                            label='Fin'
                        />
                        <Table.Column
                            sort='sortEmbarcacion'
                            label='Embarcación'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={zarpes}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={zarpes.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Zarpe({ resource, authorizations: { Create, Update, Delete } }) {
    const zarpe = resource === '~'
        ? null
        : await readZarpe(resource)

    if (zarpe || resource === '~') {
        const embarcaciones = await getArrayEmbarcaciones()

        return (
            <Container
            type='page'
        >
                <Form
                    Data={zarpe}
                    Create={Create ? createZarpe : null}
                    Update={Update ? updateZarpe : null}
                    Delete={Delete ? deleteZarpe : null}
                    pathname='/usuario/zarpes'
                    redirect='id'
                >
                    <Form.Header
                        title={`${!zarpe ? 'Crear' : Update ? 'Editar' : 'Visualizar'} zarpe`}
                    />
                    <Form.Body>
                        <Form.PrimaryKey
                            property='id'
                            label='Código del zarpe'
                        />
                        <Form.Date
                            property='inicio'
                            higher='fin'
                            required
                            label='Fecha de inicio'
                        />
                        <Form.Date
                            property='fin'
                            lower='inicio'
                            required
                            label='Fecha de fin'
                        />
                        <Form.Autocomplete
                            property='embarcacion'
                            options={embarcaciones}
                            required
                            label='Embarcación del zarpe'
                        />
                    </Form.Body>
                </Form>
            </Container>
        )
    } else {
        return <Info message={`No se encontró el zarpe con código \'${resource}\'`} />
    }
}
```

Anexo 13, Código del requerimiento funcional F11 Gestión de ventas.

```
export async function Ventas({ searchParams, authorizations: { Create } }) {
    const ventas = await getVentas(searchParams)
    const embarcaciones = await getArrayEmbarcaciones()

    return (
        <Container
            type='page'
        >
            <Search>
                <Search.Header
                    title='Ventas'
                    Create={Create}
                />
                <Search.Body>
                    <Search.String
                        property='id'
                        label='Número de la factura'
                    />
                    <Search.Date
                        property='inicio'
                        max='fin'
                        label='Fecha desde'
                    />
                    <Search.Date
                        property='fin'
                        min='inicio'
                        label='Fecha hasta'
                    />
                    <Search.Number
                        property='minGalones'
                        type='lgInt'
                        max='maxGalones'
                        label='Galones vendidos desde'
                    />
                    <Search.Number
                        property='maxGalones'
                        type='lgInt'
                        min='minGalones'
                        label='Galones vendidos hasta'
                    />
                    <Search.Autocomplete
                        property='embarcacion'
                        options={embarcaciones}
                        label='Embarcación'
                    />
                </Search.Body>
            </Search>
            <Table>
                <Table.Container>
                    <Table.Head>
                        <Table.Column
                            sort='sortID'
                            label='Factura'
                        />
                        <Table.Column
                            sort='sortFecha'
                            label='Fecha'
                        />
                        <Table.Column
                            sort='sortGalones'
                            label='Galones'
                        />
                        <Table.Column
                            sort='sortEmbarcacion'
                            label='Embarcación'
                        />
                    </Table.Head>
                    <Table.Body
                        rows={ventas}
                        searchParams={searchParams}
                    />
                </Table.Container>
                <Table.Foot
                    total={ventas.length}
                />
            </Table>
        </Container>
    )
}
```

```
export async function Venta({ resource, authorizations: { Create, Update, Delete } }) {
    const venta = resource === '~'
        ? null
        : await readVenta(resource)

    if (venta || resource === '~') {
        const embarcaciones = await getArrayEmbarcaciones()

        const pagos = venta
            ? await readPagos(venta.id)
            : []
        const personas = await getArrayPersonas()
        const categorias = await getArrayCategorias('ventas')

        return (
            <Container
            type='page'
        >
                <Form
                    Data={venta}
                    Create={Create ? createVenta : null}
                    Update={Update ? updateVenta : null}
                    Delete={Delete ? deleteVenta : null}
                    pathname='/usuario/ventas'
                    redirect='id'
                >
                    <Form.Header
                        title={`${!venta ? 'Crear' : Update ? 'Editar' : 'Visualizar'} venta`}
                    />
                    <Form.Body>
                        <Form.PrimaryKey
                            property='id'
                            label='Número de la factura'
                        />
                        <Form.Date
                            property='fecha'
                            required
                            label='Fecha de la factura'
                        />
                        <Form.Number
                            property='galones'
                            type='lgInt'
                            required
                            label='Galones vendidos'
                        />
                        {
                            venta && (
                                <>
                                    <Form.Number
                                        property='total'
                                        type='lgFloat'
                                        label='Valor total'
                                        prefix='$'
                                    />
                                    <Form.Field
                                        property='pagado'
                                        label='Valor pagado'
                                        prefix='$'
                                    />
                                    <Form.Field
                                        property='restante'
                                        label='Valor restante'
                                        prefix='$'
                                    />
                                </>
                            )
                        }
                        <Form.Autocomplete
                            property='embarcacion'
                            options={embarcaciones}
                            required
                            label='Embarcación de la factura'
                        />
                        {
                            venta && (
                                <Form.Field
                                    property='zarpe'
                                    label='Zarpe de la embarcación'
                                />
                            )
                        }
                        <Form.String
                            property='observaciones'
                            area
                            label='Observaciones'
                        />
                    </Form.Body>
```

```
                    {
                        venta && (
                            <Form.Body
                                hideActions
                            >
                                <Form.Field
                                    property='usuario'
                                    label='Usuario del último cambio'
                                />
                                <Form.Field
                                    property='date'
                                    label='Fecha del último cambio'
                                />
                            </Form.Body>
                        )
                    }
                </Form>
                {
                    venta && (
                        <Multiple
                            parent={venta.current_id}
                            childs={pagos}
                            action={Update ? updatePagos : null}
                        >
                            <Multiple.Header
                                title='Pagos'
                            />
                            <Multiple.Body
                                columns={[
                                    'Valor',
                                    'Fecha',
                                    'Representante',
                                    'Método de pago'
                                ]}
                            >
                                <Multiple.Number
                                    property='valor'
                                    type='lgFloat'
                                    required
                                />
                                <Multiple.Date
                                    property='fecha'
                                    required
                                />
                                <Multiple.Autocomplete
                                    property='persona'
                                    options={personas}
                                    required
                                    notStrict
                                />
                                <Multiple.Autocomplete
                                    property='categoria'
                                    options={categorias}
                                    required
                                    notStrict
                                />
                            </Multiple.Body>
                        </Multiple>
                    )
                }
            </Container>
        )
    } else {
        return <Info message={`No se encontró la venta con código \'${resource}\'`} />
    }
}
```

Anexo 14, Código del requerimiento funcional F12 Gestión de reportes.

```
export default async function Reportes({ searchParams }) {
    const sectores = await getArrayCategorias('personas')
    const personas = await readPersonas(searchParams.sectores)

    const { registros, datos } = date.test(searchParams.inicio) || date.test(searchParams.fin)
        ? await getReporte(personas, searchParams)
        : { registros: [], datos: {} }

    const embarcaciones = registros.map(registro => ({
        value: registro.embarcacion,
        label: `${registro.matricula}, ${registro.matricula_nombre}`
    }))

    return (
        <Report
            Board={registros}
            Stats={datos}
            searchParams={searchParams}
        >
            <Report.Picker
                property='sectores'
                options={sectores}
                label='Sectores'
            />
            <Report.Picker
                property='propietarios'
                options={personas}
                label='Propietarios'
            />
            <Report.Picker
                property='tripulantes'
                options={personas}
                label='Tripulantes'
            />
            <Report.Picker
                property='embarcaciones'
                options={embarcaciones}
                label='Embarcaciones'
            />
        </Report>
    )
}
```