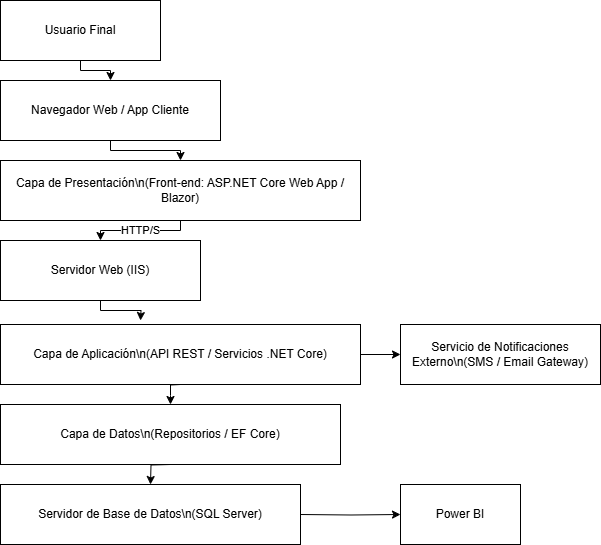
**ASSESSMENT**

*Documentación Técnica y de Proyecto*

Danilo Estuardo Itzep Luna

19/06/2025

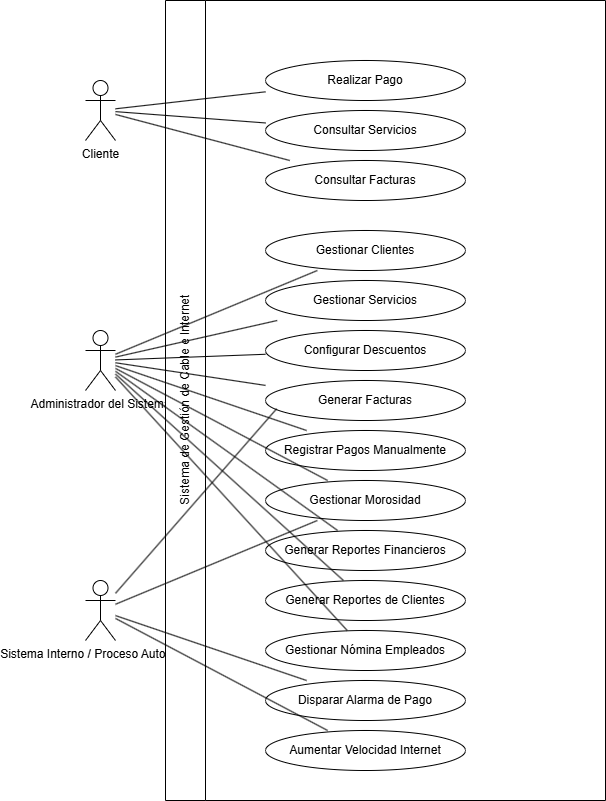
**1.Diagrama de la solución:**



El diagrama representa la arquitectura de la solución desarrollada bajo un enfoque por capas, promoviendo la separación de responsabilidades, la escalabilidad y el mantenimiento eficiente del sistema.

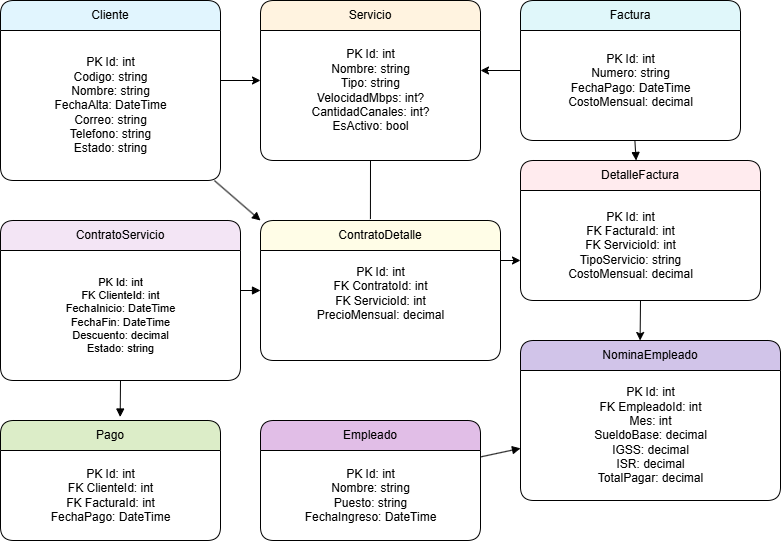
1. **Usuario Final**: Es el actor que interactúa con la solución desde un navegador web. Accede a funcionalidades como gestión de clientes, servicios y asignación de contratos.
2. **Navegador Web / App Cliente**: El acceso se realiza a través de una interfaz web desarrollada en ASP.NET Core MVC (o Blazor, si se desea extender), que se comunica con la API a través de peticiones HTTP.
3. **Capa de Presentación (Front-end)**: Es el cliente que consume los endpoints REST. Esta capa se encarga de mostrar los datos al usuario, enviar formularios y mostrar confirmaciones. Utiliza HttpClient para interactuar con la API.
4. **Servidor Web (IIS)**: Hospeda tanto la aplicación web como la API. Gestiona las peticiones entrantes y las enruta hacia los controladores correspondientes del backend.
5. **Capa de Aplicación (API REST)**: Esta es la lógica central implementada en un proyecto ASP.NET Core API. Aquí se encuentran los controladores que exponen los servicios, validan los datos y coordinan la lógica mediante servicios de dominio.
6. **Capa de Datos (EF Core / Repositorios)**: Interactúa directamente con la base de datos a través del patrón Repositorio y Unit of Work. Se utiliza Entity Framework Core para mapear entidades y ejecutar operaciones CRUD de forma segura y desacoplada.
7. **Servidor de Base de Datos (SQL Server)**: Almacena toda la información de la solución: clientes, servicios, contratos, pagos, facturas y empleados. También contiene procedimientos almacenados que pueden ser consumidos por Power BI o desde la propia API.
8. **Servicio Externo de Notificaciones (Opcional)**: Está previsto un canal de integración futura con servicios de correo, SMS o gateways de terceros para notificaciones automáticas al cliente.
9. **Power BI**: Conectado directamente a la base de datos, Power BI permite generar reportes dinámicos mediante consultas SQL y procedimientos almacenados, ofreciendo a la empresa información clave para la toma de decisiones.

**2.Diagrama UML:**



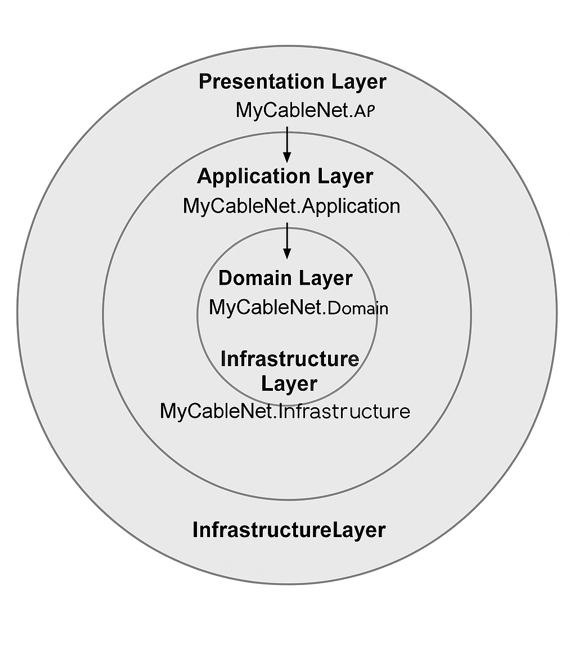
El diagrama de casos de uso describe las principales funcionalidades del sistema y los actores que interactúan con él. El **cliente** puede consultar servicios, revisar facturas y realizar pagos. El **administrador del sistema** se encarga de la gestión de clientes, servicios, facturación, descuentos, y reportes. Por su parte, el **sistema interno** realiza tareas automáticas como generar nóminas, activar alarmas por morosidad y aumentar la velocidad de internet para clientes en buen estado. Esta visión general permite identificar claramente las responsabilidades y funcionalidades ofrecidas por el sistema.

**3.Diagrama entidad relación:**



El diagrama ER representa la estructura de base de datos del sistema de gestión de servicios de cable e internet. Cada entidad refleja un módulo funcional clave: clientes, contratos, servicios, facturación, pagos y empleados. Las relaciones permiten una trazabilidad clara desde el cliente hasta sus servicios contratados, facturas generadas, pagos realizados y empleados involucrados. Este diseño garantiza integridad referencial y soporte para operaciones complejas como generación automática de facturas, alarmas por morosidad y cálculo de nómina mensual.

**Arquitectura de la solución:**

****

**MyCableNet.Domain (Núcleo de dominio)**

* **Propósito**: Contiene las entidades empresariales puras (clases con lógica de negocio), totalmente independientes.
* **No depende de nada**.
* **Ejemplo**: Cliente, Servicio, Factura, etc.

**MyCableNet.Application (Lógica de aplicación)**

* **Propósito**: Define los casos de uso del sistema.
* **Contiene**: DTOs, Interfaces, Servicios de aplicación, lógica de validación.
* **Depende solo del dominio.**
* **Ejemplo**: Servicios que coordinan operaciones como registrar pagos, consultar servicios, etc.

**MyCableNet.Infrastructure (Infraestructura)**

* **Propósito**: Implementa las interfaces definidas en la capa de aplicación.
* **Contiene**: EF Core, repositorios, DbContext, acceso a base de datos, UnitOfWork.
* **Depende de Application y Domain.**
* **Ejemplo**: ClienteRepository, MyCableNetDbContext, clases de migración.

**MyCableNet.API (Presentación / Entrada)**

* **Propósito**: Provee los endpoints que consumen los clientes (por ejemplo, navegadores o apps).
* **Contiene**: Controladores, configuración, servicios Hosted, Program.cs.
* **Depende de Application e Infrastructure.**
* **Ejemplo**: ClientesController, FacturasController.

# 4.Requerimientos Funcionales y No Funcionales

**-Requerimientos Funcionales (RF)**

Describen lo que el sistema debe hacer funcionalmente.

**1. Gestión de Clientes**

RF01. El sistema debe permitir registrar, consultar, actualizar y eliminar (lógicamente) clientes.

RF02. Los datos del cliente deben incluir: código, nombre, fecha de alta, dirección, correo, teléfono y estado.

**2. Gestión de Servicios**

RF03. El sistema debe permitir configurar (crear, modificar, consultar, eliminar lógicamente) servicios.

RF04. Debe soportar servicios de cable (básico, premium) y internet (15Mbps, 25Mbps, 50Mbps).

**3. Asignación y Contratación de Servicios**

RF05. El sistema debe permitir asignar uno o dos servicios por cliente.

RF06. El sistema debe llevar control de lugar de instalación del servicio.

**4. Gestión de Contratos**

RF07. El sistema debe registrar contratos con servicios asociados, indicando fecha de inicio, fin, estado y si es paquete doble.

**5. Gestión de Costos y Descuentos**

RF08. El sistema debe permitir configurar los precios de servicios, considerando la ubicación del cliente.

RF09. El sistema debe aplicar un 10% de descuento si el cliente contrata 2 servicios.

RF10. El sistema debe aplicar promoción: si el cliente paga 6 meses anticipados, se cobran solo 5.

RF11. El sistema debe permitir crear y gestionar tipos de descuentos (por tipo de cliente, cantidad de servicios, etc.).

**6. Facturación y Pagos**

RF12. El sistema debe generar facturas con número, serie, fecha, cliente y detalle de servicios.

RF13. El sistema debe registrar los pagos realizados por los clientes, asociados a facturas.

RF14. El sistema debe identificar a un cliente como moroso si no paga durante 2 meses consecutivos.

RF15. A partir del tercer mes de impago, debe bloquear la factura y suspender automáticamente los servicios.

RF16. El sistema debe generar una alarma de pago cuando corresponde facturar al cliente.

**7. Reportes**

RF17. El sistema debe generar reportes de:

* Clientes que han pagado en el año.
* Clientes activos, morosos y suspendidos.
* Ganancias por mes.
* Deudas por cobrar.
* Facturación detallada.

**8. Aumento automático de velocidad**

RF18. El sistema debe aumentar automáticamente la velocidad del servicio de internet en 5Mbps por año a clientes que hayan pagado puntualmente, hasta un máximo de 50Mbps.

**9. Gestión de Empleados y Nómina**

RF19. El sistema debe registrar empleados con nombre, puesto, fecha de ingreso y salario base.

RF20. El sistema debe generar la nómina mensual de empleados incluyendo:

Cálculo de IGSS, ISR

Pasivo laboral

Total a pagar

Estado de pago

**-Requerimientos No Funcionales (RNF)**

Definen cómo debe comportarse el sistema en términos de calidad, seguridad y tecnología.

**1. Rendimiento**

RNF01. Las consultas de clientes, facturación y servicios deben responder en menos de 2 segundos.

**2. Disponibilidad**

RNF02. El sistema debe estar disponible al menos el 99.5% del tiempo en horario laboral.

**3. Seguridad**

RNF03. El sistema debe implementar autenticación y autorización por roles (admin, técnico, facturación, etc.).

RNF04. La información sensible debe estar protegida contra accesos no autorizados.

RNF05. Las conexiones a base de datos deben estar cifradas y seguras.

**4. Escalabilidad**

RNF06. El sistema debe escalar horizontalmente para soportar un crecimiento de clientes y servicios sin degradar el rendimiento.

5. Mantenibilidad

RNF07. El sistema debe tener código modular, documentado y desacoplado para permitir evolución y mantenimiento.

RNF08. Los precios, descuentos y tipos de servicios deben ser configurables sin modificar el código.

**6. Fiabilidad y robustez**

RNF09. El sistema debe manejar errores de forma controlada, registrarlos, y permitir recuperación sin pérdida de datos.

**7. Usabilidad**

RNF10. La interfaz debe ser intuitiva, con pantallas CRUD (mantenimiento) claras y validaciones visibles para el usuario.

**8. Compatibilidad**

RNF11. El sistema debe funcionar correctamente en navegadores web modernos (Chrome, Firefox, Edge) o versiones actuales de Windows si es escritorio.

**9. Tecnología**

RNF12. El sistema debe desarrollarse en .NET y utilizar SQL Server como motor de base de datos.

**10. Reportabilidad**

RNF13. La estructura de datos debe estar optimizada para herramientas como Power BI, permitiendo generación de dashboards y reportes avanzados.

# 5.Plan de Trabajo Propuesto

**Plan de Trabajo Propuesto (Iterativo Ágil con Scrum)**

Metodología: Ágil (Scrum)

Ciclos de trabajo organizados en Sprints de 2 a 3 semanas

Entregas incrementales, revisiones periódicas, adaptabilidad a cambios

IDE: Visual Studio 2022+, SQL Server Management Studio

DB: SQL Server

Control de versiones: Git (Azure DevOps o GitHub)

Gestión: Jira

Infraestructura: Azure App Services + SQL Azure o VMs

Reporting: Power BI (Desktop + Servicio)

**Fases del Proyecto (4 fases con duración estimada: 4 a 6 meses)**

**Fase 1:** Planificación y Descubrimiento (2-3 semanas)

Actividades:

Reunión de arranque (kick-off)

Levantamiento y validación de requisitos

Priorización del Product Backlog (Historias de Usuario)

Diseño de arquitectura y diagrama entidad-relación (ER)

Configuración de entorno de desarrollo, DevOps y versionado

Entregables:

Backlog inicial validado

Diagrama de arquitectura y ER

Ambiente de desarrollo y control de versiones configurado

Recursos: Product Owner, Analista, Scrum Master, Devs

**Fase 2:** Desarrollo del Core (8-12 semanas)

Actividades:

Desarrollo iterativo (sprints)

CRUDs de Clientes, Servicios, Contratos, Pagos

Lógica de asignación de servicios

Primeras validaciones de descuentos fijos

Base de datos inicial con procedimientos (SPs)

Pruebas unitarias automatizadas y manuales

Entregables:

Módulos funcionales: Clientes, Servicios, Contratos

Validación de reglas básicas de negocio

BD conectada y funcional

Recursos: Devs, QA, Scrum Master, PO (feedback)

**Fase 3:** Lógica avanzada y Reportes (8-10 semanas)

Actividades:

Lógica avanzada:

Descuentos configurables

Validación y gestión de morosidad

Aumento automático de velocidad

Implementación del módulo de nómina (IGSS, ISR, pasivo laboral)

Alarmas automáticas de pago

Desarrollo de reportes en Power BI

Consultas SQL avanzadas para visualización

Entregables:

Reglas de negocio completas operativas

Módulo de Nómina funcional

Reportes dinámicos Power BI conectados

Recursos: Devs, QA, Product Owner, BI Developer

**Fase 4:** Pruebas Finales, Capacitación y Despliegue (3-4 semanas)

Actividades:

Pruebas funcionales, de regresión, rendimiento y seguridad

Corrección de bugs y estabilización

Preparación de documentación técnica y manual de usuario

Capacitación a usuarios clave

Despliegue a producción y monitoreo post-go live

Entregables:

Sistema completamente probado y desplegado

Manuales técnicos y funcionales

Usuarios capacitados

Sistema en producción

Recursos: QA, Devs, PO, DevOps

Consideraciones Finales

Tiempos aproximados:

Buenas Prácticas y Riesgos

Buffer del 10-20% para imprevistos.

Daily Meetings, Sprint Reviews y Retrospectives.

Código desacoplado por capas (Clean Architecture).

Seguridad desde el inicio (roles, validaciones).

Riesgos: cambios tardíos en requisitos, carga de reportes pesados sin optimizar consultas.

# 6.Especificación de Requerimientos (DERCAS)

Puntos Clave que Debe Incluir un Documento de Especificación de Requerimientos de Desarrollo (DERCAS)

El DERCAS (Documento de Especificación de Requerimientos para el Desarrollo de Software) es un instrumento esencial que describe qué se va a construir, por qué, cómo debe comportarse y cuáles son los límites técnicos y funcionales del sistema.

**1. Introducción**

Propósito del Documento: Explica qué problema resuelve el software y para quién se desarrolla.

Alcance del Sistema: Define las funcionalidades incluidas y excluidas.

Audiencia Objetivo: Especifica a quién va dirigido (PO, usuarios, devs, QA, dirección).

Definiciones y Acrónimos: Glosario de términos técnicos o de negocio utilizados.

**2. Descripción General del Producto**

Perspectiva del Producto: Relación con otros sistemas, integración con servicios externos (ej. Power BI, email, etc.).

Resumen de Funcionalidad: Visión de alto nivel de las funciones principales.

Tipos de Usuarios y Roles: Perfil de los usuarios (cliente, administrador, técnico, facturador, etc.).

Restricciones:

Tecnológicas (lenguaje, framework, sistema operativo).

Organizacionales (presupuesto, equipo disponible).

Regulatorias (protección de datos, cumplimiento legal).

Suposiciones y Dependencias: Consideraciones necesarias para el desarrollo (ej. disponibilidad de base de datos, servicios externos).

**3. Requerimientos Funcionales Detallados**

Para cada funcionalidad:

ID Único (ej. RF01)

Nombre del Requerimiento

Descripción Detallada: Qué hace, cómo funciona, y qué espera el usuario.

Actores Involucrados: Usuarios o sistemas que interactúan.

Prioridad (Alta, Media, Baja)

Precondiciones: Estado previo necesario.

Flujo Básico: Pasos normales para cumplir el requerimiento.

Flujos Alternativos y Excepciones: Qué pasa si algo sale mal.

Reglas de Negocio Asociadas:

Aplicación de descuentos.

Morosidad.

Suspensión de servicios.

Lógica de facturación.

Criterios de Aceptación: Qué condiciones deben cumplirse para considerar implementado el requerimiento.

**4. Requerimientos No Funcionales**

Rendimiento: Tiempo de respuesta, cantidad máxima de usuarios simultáneos.

Seguridad: Roles, autenticación, autorización, cifrado, trazabilidad.

Usabilidad: Facilidad de uso, accesibilidad, navegación intuitiva.

Fiabilidad: Recuperación ante fallos, disponibilidad, redundancia.

Mantenibilidad: Modularidad, documentación, facilidad de refactorización.

Escalabilidad: Capacidad de manejar más usuarios, datos o procesos sin rediseño.

Compatibilidad: Navegadores soportados, plataformas (Windows, Web).

Portabilidad: Posibilidad de migrar entre entornos (Dev, Test, Prod).

Tecnología: Tecnologías requeridas (.NET 8, SQL Server, Power BI, etc.).

Regulatorios: Cumplimiento de normativas específicas (si aplica).

**5. Requerimientos de Interfaces**

Interfaces de Usuario (UI):

Wireframes, mockups, prototipos.

Comportamiento esperado de cada pantalla.

Interfaces de Software (API):

Definición de servicios REST, entrada/salida esperada.

Conexión con servicios externos (notificaciones, Power BI, servicios contables).

Interfaces de Hardware (si aplica): Impresoras, sensores, etc.

Interfaces de Comunicación:

Protocolos utilizados (HTTP, HTTPS, FTP, SMTP).

Seguridad en transporte de datos.

**6. Requerimientos de Base de Datos**

Modelo Entidad-Relación: Diagrama ER validado.

Esquema Relacional: Tablas, columnas, PK, FK, tipos de datos.

Consultas Críticas: Ej. para reportes, Power BI, auditorías.

Procedimientos Almacenados / Vistas / Funciones requeridas.

**7. Consideraciones de Diseño Técnico**

Patrones de Diseño a Utilizar: MVC, Repositorio, Unit of Work, CQRS, etc.

Arquitectura Propuesta: Capas (Presentation, Application, Infrastructure, Domain).

Buenas Prácticas y Estándares de Codificación: Naming conventions, manejo de errores, logs, etc.

**8. Apéndices**

Diagramas UML (casos de uso, clases, secuencia).

Diagrama de Arquitectura.

Referencias (caso de estudio, normas, documentación de API).

Plantillas o formatos usados (ej. historia de usuario, criterios de aceptación).

Glosario técnico/funcional extendido.

Resultado Esperado

Un DERCAS con esta estructura:

Elimina ambigüedades

Facilita la trazabilidad entre requerimientos y funcionalidades

Sirve como contrato funcional y técnico entre los stakeholders y el equipo de desarrollo

Reduce los errores de interpretación durante el desarrollo y pruebas

# 7.Cómo Apoya DevOps en el Proyecto

¿Cómo apoya DevOps en la gestión de los proyectos o casos asignados? ¿Y por qué?

DevOps es una cultura y conjunto de prácticas que integran a los equipos de Desarrollo (Dev) y Operaciones (Ops) para lograr entregas más rápidas, confiables, automatizadas y de mayor calidad. Su implementación en este proyecto aporta valor transversal en todo el ciclo de vida del software.

**1. Integración y Entrega Continua (CI/CD)**

Apoyo: Automatiza la construcción, pruebas y despliegue del sistema (por ejemplo, backend en .NET y base de datos SQL Server).

Por qué:

Reduce errores humanos.

Asegura que cada cambio se pruebe y esté listo para desplegarse.

Permite responder rápido ante solicitudes de mejora o corrección (por ejemplo, nuevos descuentos o lógica de morosidad).

Facilita la entrega frecuente de funcionalidades en ciclos cortos (iteraciones).

**2. Automatización de Pruebas**

Apoyo: Permite integrar pruebas unitarias, de integración y funcionales dentro del pipeline DevOps.

Por qué:

Garantiza que funcionalidades críticas (como descuentos, bloqueo por morosidad, velocidad de internet) funcionen como se espera antes de llegar al entorno productivo.

Disminuye costos de corrección al detectar errores temprano.

**3. Monitoreo y Observabilidad**

Apoyo: Herramientas como Azure Monitor, Application Insights, o Prometheus para observar rendimiento y errores.

Por qué:

Detecta caídas del sistema o lentitud en módulos (como reportes, alarmas o pagos).

Permite actuar de forma proactiva ante fallos.

Mide el uso de recursos (CPU, RAM, queries lentas de SQL Server).

**4. Gestión de Configuración**

Apoyo: Define parámetros del sistema como costos de servicios, descuentos, rutas, claves en archivos versionados (por ejemplo, appsettings.json, Azure Key Vault).

Por qué:

Evita diferencias entre ambientes (dev, test, prod).

Facilita rollback de versiones anteriores.

Permite trazabilidad total.

**5. Colaboración y Transparencia**

Apoyo: DevOps fomenta comunicación constante entre desarrollo, QA y operaciones a través de herramientas como:

Azure DevOps Boards

Git (Repos)

Revisión colaborativa de código (pull requests)

Por qué:

Evita silos.

Acelera la resolución de problemas.

Mejora la planificación y coordinación en equipos multifuncionales.

**6. Control de Versiones y Gestión de Cambios**

Apoyo: Uso de Git para versionar código fuente, scripts de base de datos, configuraciones, pruebas y pipelines.

Por qué:

Permite trazabilidad y auditoría.

Facilita rollback ante errores.

Mejora la colaboración entre múltiples desarrolladores.

**7. Seguridad Integrada (DevSecOps)**

Apoyo: Integración de escaneos de vulnerabilidades, pruebas de seguridad automatizadas y control de acceso a repositorios e infraestructura.

Por qué:

Protege la información sensible (datos de clientes, empleados).

Refuerza políticas de cumplimiento y estándares de calidad.

# 8.Propuesta de Aseguramiento de la Calidad

Propuesta de Manejo para el Aseguramiento de la Calidad del Proyecto

El aseguramiento de la calidad (QA) debe integrarse desde el inicio del ciclo de vida del proyecto y mantenerse de forma continua en cada fase de desarrollo. Abarca no solo verificación técnica, sino también alineación con los objetivos del negocio, prevención de errores y satisfacción del usuario final.

**1. Planificación de la Calidad**

Definición de Criterios de Calidad:

Tiempo de respuesta máximo aceptado (ej. < 2 segundos para consultas críticas).

Tasa máxima de defectos por módulo (< 3%).

Cobertura mínima de pruebas (> 80% en módulos críticos).

Revisión de Requerimientos:

Validar que los requerimientos funcionales y no funcionales sean claros, completos y medibles (ej. lógica de descuentos, reglas de morosidad, aumentos automáticos de velocidad).

Plan de Pruebas:

Establecer tipos de pruebas por fase.

Asignación de responsables (QA, desarrolladores, negocio).

Herramientas a utilizar (Azure DevOps, Selenium, Postman, JMeter).

Métricas de Seguimiento:

Defectos por sprint.

Defectos reabiertos.

Tiempo promedio de resolución.

Velocidad de entrega de historias con criterios de calidad completos.

**2. Calidad en el Desarrollo**

Revisiones de Código (Code Reviews):

Validar estándares de codificación (.NET, SQL), seguridad, y legibilidad.

Verificar principios SOLID, uso correcto de patrones de diseño, separación de capas, y SRP.

Pruebas Unitarias Automatizadas:

Realizadas por desarrolladores para validar funciones aisladas como: cálculo de descuentos, reglas de morosidad, validación de fechas de corte.

Pruebas de Integración:

Validar que los módulos se comuniquen correctamente: SPs, APIs internas, lógica de facturación.

Control de Ramas y Pull Requests:

Establecer políticas de integración segura y controlada a través de Git y Azure DevOps Repos.

**3. Pruebas Formales e Iterativas**

Pruebas Funcionales (por Sprint):

Manuales para flujos de negocio complejos.

Automatizadas (Selenium para UI, Postman/Newman para APIs).

Pruebas de Regresión:

Ejecutadas en cada Sprint para verificar que nuevas funcionalidades no rompan lo existente.

Pruebas de Rendimiento (Load y Stress Testing):

Evaluar cuántas facturas se pueden procesar por minuto.

Asegurar que reportes (ej. Power BI) se generen en < X segundos bajo carga.

Pruebas de Seguridad:

Escaneo de vulnerabilidades con herramientas como OWASP ZAP.

Validación contra ataques comunes: SQL Injection, manejo de sesiones.

Pruebas de Usabilidad:

Feedback de usuarios sobre pantallas de mantenimiento (ABC).

Ajustes según criterios de accesibilidad y experiencia de usuario.

Pruebas de Aceptación del Usuario (UAT):

Validación por parte del cliente final.

Validación de módulos críticos como nómina, facturación, reportes.

**4. Gestión de Defectos y Calidad**

Proceso de Defectos:

Estados: Abierto → Asignado → En progreso → Validado → Cerrado.

Priorización: Crítico, Alto, Medio, Bajo.

Herramientas: Azure Boards, Jira.

Control de Entornos de Prueba:

Separación de ambientes: dev / test / staging / producción.

Datos de Prueba:

Datos que representen distintos escenarios:

Cliente con 1 servicio.

Cliente moroso.

Cliente con 6 pagos adelantados.

Clientes con aumento automático de velocidad.

**5. Mejora Continua**

Retrospectivas QA:

Evaluar por sprint qué prácticas funcionaron.

Identificar cuellos de botella en pruebas o errores recurrentes.

Análisis de Causa Raíz:

Documentar y prevenir con checklist o pruebas adicionales.

Capacitación Continua:

Talleres de calidad de software, buenas prácticas, herramientas de testing.