ChatGPT

Nordia QA Standard v1.0 – ChatGPT Zenith QA

Versión: 07 de diciembre de 2025 – alineado con la arquitectura de 4 motores de Nordia Quality Factory (Analyzer, Inspector, Pilot y Brain)

Introducción

Nordia POS es un sistema de punto de venta para carnicerías, verdulerías y kioscos que aspira a competir con los grandes sin perder sencillez. La calidad de este software no puede ser un "extra": en un comercio real, cada minuto de falla implica ventas perdidas y clientes molestos. Este estándar oficial de QA busca que **todos** –desarrolladores, testers, product managers y founders– compartan la misma definición de calidad, conozcan el flujo de verificación y tengan herramientas concretas para medir y mejorar.

La norma se inspira en el modelo de calidad **ISO/IEC 25010**, que define características como confiabilidad, seguridad y mantenibilidad. Según ISO 25010, la **confiabilidad** es la capacidad de un sistema para cumplir funciones especificadas bajo condiciones definidas durante un periodo de tiempo; incluye subcaracterísticas como disponibilidad, tolerancia a fallas y recuperabilidad [1]. La **mantenibilidad** se refiere a la eficacia y eficiencia con que un producto puede modificarse para corregir defectos o adaptarse a cambios; abarca modularidad, reutilización, analizabilidad, modificabilidad y testabilidad [2]. Estas definiciones son el marco teórico de este estándar.

El estándar se divide en diez secciones:

1. **Gobernanza y roles de QA** 2. **Matriz de severidad y prioridad de bugs** 3. **Flujo de QA y criterios de salida** 4. **Plantilla JSON de bug** 5. **Plantilla JSON de caso de prueba** 6. **Lista de Smoke Tests** 7. **Checklists por vertical (carnicería, verdulería, kiosco)** 8. **Escenarios rápidos para el QA Agent** 9. **Métricas y KPIs de calidad** 10. **Contrato de liberación (QA Release Contract)**

Cada sección incluye definiciones, procesos y ejemplos adaptados al contexto de Nordia.

---

1. Gobernanza y roles de QA

Nordia Quality Factory adopta un modelo **multicapa**, donde cada rol tiene responsabilidades claras y existe un "gate" de aprobación entre cada capa.

Estructura de roles

Rol	Responsabilidades principales
Founder & QA Lead (Gonzalo)	Define la visión de negocio y prioriza; realiza testing manual experto; aprueba el contrato de liberación.

Rol	Responsabilidades principales
Director Técnico QA (Claude Opus 4.5)	Implementa la arquitectura de 4 motores; desarrolla y mantiene el QA Agent, integra Playwright, Supabase y otros servicios; ejecuta pruebas automatizadas; entrega evidencias.
Auditor Externo QA (ChatGPT Zenith)	Diseña estándares y procesos; revisa documentación y artefactos; ofrece segunda opinión independiente; ayuda a priorizar bugs y define KPIs.
Equipo de Desarrollo	Implementa funcionalidades; corrige defectos; escribe casos de prueba unitarios; participa en revisiones de código.
Usuarios Piloto	Comercios reales que prueban el producto y reportan defectos; su feedback tiene prioridad sobre la estética.

Flujo de QA

El flujo completo se resume en los siguientes pasos. Cada paso tiene un "gate" que debe aprobarse antes de avanzar:

1. **Diseño y especificación**: se definen historias de usuario y criterios de aceptación; el Auditor QA revisa que estén completos y que cubran flujos de carnicerías/verdulerías/kioscos.
2. **Desarrollo**: los desarrolladores codifican la funcionalidad y escriben pruebas unitarias. Todo código pasa por revisión ("code review") obligatoria.
3. **Analyzer (motor 1)**: el QA Agent realiza un análisis estático y contextual (prompt engineering) sobre la nueva funcionalidad. Se genera un informe preliminar de riesgos.
4. **Inspector (motor 2)**: se ejecutan pruebas dinámicas automáticas con Playwright (tests de humo y de regresión). Se capturan errores de consola, tiempos de carga y capturas de pantalla.
5. **Pilot (motor 3)**: los escenarios generados por IA se ejecutan en un entorno aislado. Si la IA detiene el flujo por inconsistencias, se revisa manualmente y se ajustan los scripts.
6. **Testing manual**: el Founder & QA Lead realiza una pasada completa de smoke tests, checklists de verticales y casos de uso críticos que no pueden automatizarse (peso en balanza, corte de caja, recarga de posnet, etc.).
7. **Pre-deploy**: se revisa la matriz de severidad; si existen fallas críticas o altas sin resolver, se **bloquea la liberación**. Se completan todos los campos del **Contrato de liberación**.
8. **Deploy controlado**: la versión se instala en los 5 comercios piloto. Se monitorean métricas de error en tiempo real (SLIs y KPIs).
9. **Post-mortem**: cualquier incidente se documenta con plantilla de bug; se actualizan las métricas (MTTR, MTBF, etc.) y se revisan acciones de mejora.

---

2. Matriz de severidad y prioridad

Niveles de severidad (defectos)

Estos niveles definen el **impacto técnico** de un defecto. Tomamos como referencia una guía de defectos: los errores de severidad **Crítica** son aquellos que causan fallo del sistema o pérdida de datos y bloquean el uso [3]; los errores **Altos** rompen funcionalidades mayores pero permiten que algunas partes sigan funcionando [4]; los errores **Medios** afectan funciones secundarias [5]; y los **Bajos** son problemas cosméticos o menores [6]. A continuación se detallan niveles y ejemplos.

Severidad	Definición (impacto)	Ejemplos
Crítica	El sistema se bloquea, corrompe datos o impide cualquier operación; no existe workaround.	Crash al iniciar; venta que borra inventario; pérdida de totales de caja.
Alta	Funcionalidad principal falla pero el sistema sigue funcionando parcialmente; existe un workaround complejo.	No se puede cobrar con tarjeta, pero sí con efectivo; balanza descalibrada en la app; incapacidad de imprimir tickets.
Media	Problema en funcionalidades secundarias que no impactan el flujo principal; se puede diferir la corrección.	Error en filtros de búsqueda; problema al exportar reportes; lentitud en la sección de configuración.
Baja	Defectos cosméticos o triviales que no afectan la operativa ni la percepción de fiabilidad.	Texto desalineado; icono incorrecto; tipografía inconsistente.

Niveles de prioridad

La **prioridad** refleja la urgencia de la corrección desde el negocio. Una vez asignada la severidad, el equipo decide la prioridad evaluando factores como número de comercios afectados, frecuencia de ocurrencia y reputación. Por ejemplo:

- **Urgente**: debe corregirse antes de cualquier release. Incluye todas las severidades críticas y algunas altas.
- **Alta**: se corrige en la próxima iteración; incluye severidades altas que afectan a pocos comercios y algunas medias que perjudican procesos clave.
- **Media**: se planifica su corrección en el backlog; generalmente severidades medias o bajas con impacto moderado.
- **Baja**: se corrige cuando haya disponibilidad; severidades bajas o solicitudes de mejora.

Este estándar requiere que cualquier bug **Crítico** o **Urgente** bloquee la liberación hasta ser resuelto o se documente un workaround aceptado por el Founder.

---

3. Flujo de QA y criterios de salida

El flujo descrito en la sección 1 debe cumplirse de manera estricta. A continuación se detallan los **criterios de salida** (exit criteria) que deben cumplirse para pasar de una fase a otra:

1. **Fin de desarrollo**: código pasó revisión; pruebas unitarias cubren al menos el 80 % de las ramas lógicas; no hay errores de compilación ni warnings graves.
2. **Analyzer OK**: el QA Agent no detecta problemas de sanitización, injections ni limitaciones de código; los prompts están dentro de los límites de tokens; se generan plantillas de bug y de test correctamente.
3. **Inspector OK**: todas las pruebas automatizadas de smoke/regresión pasan; no hay errores de consola ni requests fallidas; el tiempo de carga inicial cumple con el SLO definido (p.ej., < 3 s para la vista de ventas).
4. **Pilot OK**: los escenarios generados por IA se ejecutan sin bloqueos. Si hay pausas por falta de coincidencia, se ajustan los scripts y se repite. Las validaciones automáticas de stock y totales se comparan con datos reales de Supabase.

5. **Smoke tests manuales OK**: se han ejecutado todos los tests de la lista del apartado 6 y se han documentado los resultados. No se admiten fallos de severidad crítica ni alta.
6. **Revisión de severidad**: todos los defectos descubiertos se registran con la plantilla de bug; se asigna severidad y prioridad; no quedan defectos críticos ni urgentes abiertos.
7. **Contrato de liberación firmado**: el Founder & QA Lead y el Director Técnico QA firman el documento de liberación (sección 10) indicando que se cumplen los criterios anteriores.

Si alguno de estos criterios no se cumple, se retorna a la fase anterior hasta resolverlo. **No se permiten "bypass"** sin justificación y aprobación explícita del Founder.

---

4. Plantilla JSON de bug

Todos los defectos deben registrarse con la siguiente estructura. Esto facilita la ingestión por el QA Agent y la sincronización con herramientas como GitHub Issues.

```
{
  "id": "BUG-0001",                    // identificador único
  "title": "Error en cálculo de totales", // resumen breve del problema
  "fecha": "2025-12-07",               // fecha de detección
  "comercio": "Carnicería El Tío",       // comercio y vertical
  "steps_to_reproduce": [                // pasos exactos para reproducir
    "Ingresar al POS con usuario X",
    "Agregar producto 'Costilla' con peso 2,5 kg",
    "Aplicar descuento manual de 10%",
    "Finalizar venta"
  ],
  "expected_result": "El total debería ser (precio x peso) - 10%",  //
resultado esperado
  "actual_result": "El descuento se aplica dos veces",            //
resultado observado
  "severity": "Alta",                     // nivel de severidad (Crítica/
Alta/Media/Baja)
  "priority": "Urgente",                 // prioridad de arreglo (Urgente/
Alta/Media/Baja)
  "impact": "Afecta margen de ganancia",    // impacto en negocio
  "evidence": {
    "screenshot": "url o ruta de captura",  // evidencia visual
    "logs": "error log del navegador"
  },
  "environment": {
    "app_version": "v1.3.0",             // versión del POS
    "device": "Tablet Samsung A7",        // hardware
    "os": "Android 11",                    // sistema operativo
    "browser": "Chrome 120"
  },
  "reporter": "Nombre de quien reporta",  // puede ser tester o usuario
  "notes_for_dev": "Cálculo en archivo totals.js línea 54", // pistas para
dev
  "status": "Open"                       // Open, In Progress, Resolved,
```

```
Closed
}
```

Guía de uso

- El campo `steps_to_reproduce` debe ser tan detallado como sea necesario para que cualquier tester pueda replicar el fallo.
- `impact` debe explicar por qué el defecto es relevante desde el negocio (pérdida de ingresos, mala experiencia, incumplimiento legal, etc.).
- `evidence` debe incluir capturas de pantalla, videos o logs que demuestren el problema. Sin evidencia, el bug no será aceptado.
- `notes_for_dev` puede incluir referencias al código, sugerencias de solución o links a commits. Debe ser breve y no sustituye a la revisión de código.

---

5. Plantilla JSON de caso de prueba

Los casos de prueba definen cómo verificar que una funcionalidad cumpla los requisitos. Esta plantilla se usa tanto para pruebas manuales como automatizadas (Analyzer/Inspector/Pilot).

```
{
  "id": "TC-0001",                 // identificador único del test case
  "title": "Venta simple con efectivo", // nombre descriptivo
  "objetivo": "Verificar que el POS permita registrar una venta simple con
pago en efectivo", // propósito
  "preconditions": [
    "Usuario autenticado con rol cajero",
    "Inventario con al menos un producto disponible"
  ],
  "steps": [
    { "accion": "Buscar producto", "detalle": "Escanear código o ingresar
nombre" },
    { "accion": "Agregar al carro", "detalle": "Seleccionar cantidad/peso" },
    { "accion": "Finalizar", "detalle": "Elegir 'Efectivo' como método de
pago" }
  ],
  "expected_results": [
    "Producto aparece en el carrito con precio y cantidad correctos",
    "El total se calcula correctamente",
    "Se genera ticket con número de operación",
    "Se actualiza el stock"
  ],
  "severity": "Alta",            // severidad del flujo si falla
  "priority": "Urgente",          // prioridad de ejecución
  "tags": ["carnicería", "venta", "smoke"], // etiquetas para filtrado
  "data_sets": {
    "productos": [
      { "nombre": "Costilla", "precio_por_kg": 900 },
      { "nombre": "Chorizo", "precio_por_kg": 800 }
```

```
    ]
  },
  "environment": {
    "app_version": "v1.3.0",
    "device": "Tablet Samsung A7",
    "os": "Android 11",
    "browser": "Chrome 120"
  },
  "author": "Nombre del autor",
  "last_updated": "2025-12-07"
}
```

Guía de uso

- Los campos `preconditions` y `data_sets` permiten preparar el entorno (inventario, usuarios, configuraciones) para que el test sea reproducible.
- `tags` facilitan el filtrado por vertical, tipo de prueba (smoke, regresión, accesibilidad), prioridad o componente.
- Las `expected_results` deben ser observables; si es necesario validar cálculos, se incluyen fórmulas o valores exactos.

---

6. Lista de Smoke Tests (obligatorios)

Los siguientes 10 tests se consideran **mínimos indispensables**. Cualquier build debe pasarlos al 100 % antes de ser liberado a los comercios piloto. Se ejecutan de forma automatizada y manual.

1. **Inicio de sesión**: autenticación con credenciales válidas e invalidas; bloqueo tras intentos fallidos; recuperación de contraseña.
2. **Venta simple (efectivo)**: agregar productos, calcular total, cobrar en efectivo, imprimir ticket, actualizar stock.
3. **Venta con balanza**: ingresar peso desde balanza física; verificar conversión de unidades; calcular total y actualizar stock.
4. **Venta con tarjeta**: seleccionar método de pago; integración con pin pad; manejo de respuesta aprobada/denegada; impresión de voucher.
5. **Anulación de venta**: anular ticket emitido; reversión de stock y totales; registro en bitácora.
6. **Corte de caja**: cierre de turno; cálculo de ingresos por método de pago; detección de diferencias; exportación de reporte.
7. **Gestión de stock**: alta y edición de productos; ajuste de inventario; alertas por bajo stock.
8. **Actualización de precios**: cambiar precios masivamente; aplicación de descuentos por categoría; verificación en ventas posteriores.
9. **Reporte diario**: generación de reporte de ventas; filtros por fecha y usuario; exportación a PDF/Excel.
10. **Recuperación tras caída**: simular corte de energía o internet; reanudar operación sin pérdida de datos; sincronización de ventas offline.

Estos tests se definen como casos de prueba (sección 5) y se etiquetan con `"smoke"` para su ejecución prioritaria.

---

7. Checklists por vertical

Cada vertical tiene particularidades. Estas listas ayudan a que los testers (humanos y agentes) no olviden aspectos críticos.

Carnicería

- **Productos por peso**: calibración de balanza; conversión kg/g; tolerancias de error.
- **Desperdicio y recortes**: posibilidad de registrar merma; ajuste automático del stock.
- **Etiquetas sanitarias**: impresión de etiquetas con fecha de corte y vencimiento.
- **Promociones por combo**: carne + achuras a precio especial; cálculo correcto.
- **Control de temperatura**: registro de temperatura de cámaras para auditorías.

Verdulería

- **Múltiples unidades**: manejar productos por unidad (manzana) y por peso (banana); mezcla en la misma venta.
- **Productos perecederos**: alertas de vencimiento; descuentos automáticos en productos próximos a vencer.
- **Basura orgánica**: registro de merma; seguimiento de desperdicio.
- **Variación de precios por temporada**: reglas para aplicar precios según estación o día (ofertas de fin de semana).
- **Integración con proveedores**: recepción de nuevas partidas; actualización automática de stock.

Kiosco

- **Venta de prepago/recargas**: integración con servicios de recarga de celulares y transporte; confirmación de transacción.
- **Venta de cigarrillos**: verificación de edad; control de stock legal.
- **Promociones de golosinas**: descuentos por cantidad; regalos promocionales.
- **Productos con impuesto adicional**: cálculo de impuestos específicos (v.g. bebidas alcohólicas si se integran en el futuro).
- **Control de caja chica**: seguimiento de gastos menores (servicios, insumos) y su impacto en el corte de caja.

---

8. Escenarios rápidos para el QA Agent

Los “escenarios rápidos” son prompts predefinidos que el QA Agent puede ejecutar sin intervención humana. Cada escenario describe un flujo breve pero representativo. A continuación se proponen 20 escenarios iniciales (pueden ampliarse):

1. **Venta con balanza + efectivo**
2. **Venta con balanza + tarjeta**
3. **Venta con descuento manual**
4. **Venta de combo promocional**
5. **Anulación parcial de venta**
6. **Cambio de precio durante venta**
7. **Aplicación de impuesto adicional**
8. **Fallo de conexión en medio de venta**
9. **Recuperación de venta offline**

10. **Corte de caja con diferencias**
11. **Ingreso de nuevo producto (carnicería)**
12. **Ingreso de nuevo producto (verdulería)**
13. **Recarga de celular**
14. **Venta con error en balanza**
15. **Cierre de sesión y reapertura**
16. **Exportación de reporte de ventas**
17. **Edición de inventario con múltiple unidad**
18. **Promoción de último momento**
19. **Fallo en impresión de ticket**
20. **Verificación de etiquetas sanitarias**

Cada uno de estos escenarios se traduce a pasos automatizados y se guarda como caso de prueba (ver plantilla). El QA Agent debe permitir ejecutar escenarios individuales y cadenas de escenarios.

---

9. Métricas y KPIs de calidad

Para medir el éxito del proceso de QA se definen indicadores cuantitativos. Algunas métricas se basan en prácticas de **Site Reliability Engineering (SRE)** y se adaptan al contexto de Nordia.

Indicadores de confiabilidad y disponibilidad

- **SLA/SLO/SLI**: Un **SLO** es un objetivo numérico para un indicador (por ejemplo, 99 % de ventas procesadas en menos de 2 segundos). Un **SLI** es la métrica que mide ese objetivo (porcentaje de ventas que cumplen el tiempo) y un **SLA** es el acuerdo formal que incluye los SLOs y las consecuencias si no se cumplen [7]. Ejemplo: SLO 1 – "El tiempo de respuesta desde que se escanea un producto hasta que aparece en pantalla debe ser $\leq$ 500 ms en el 95 % de los casos durante horas pico"; SLI 1 – proporción de transacciones que cumplen ese tiempo.
- **MTBF (Mean Time Between Failures)**: tiempo promedio entre fallas de un componente reparable; se calcula sumando las vidas útiles y dividiéndolas por el número de fallas [8]. Permite estimar cuántas horas de operación se obtienen antes de una falla.
- **MTTR (Mean Time To Repair/Restore)**: tiempo promedio para reparar un sistema una vez detectada la falla. Según la guía, se calcula dividiendo el tiempo total de reparación entre el número de reparaciones [9]. En un POS, se mide desde que se reporta un incidente hasta que se vuelve a operar normalmente.
- **MTRS (Mean Time to Restore Service)**: promedio del tiempo necesario para recuperar el servicio completo (incluyendo detección y reparación) [10]. Permite evaluar la rapidez global del equipo.
- **MTBSI (Mean Time Between Service Incidents)**: suma de MTBF y MTRS; indica el promedio de horas entre incidentes [11].
- **MTTD (Mean Time to Detect)**: tiempo promedio en detectar un incidente [12]. En Nordia, se refiere a cuánto tarda el sistema o el comercio en darse cuenta de un fallo (p.ej., error de stock).
- **MTTI (Mean Time to Identify)**: tiempo promedio para identificar la causa raíz de un problema [13].

Indicadores de defectos y calidad de software

- **Defect density**: número de defectos por función o por líneas de código; ayuda a identificar componentes problemáticos.

- **Defect escape rate**: porcentaje de defectos que se encuentran en producción en comparación con los detectados en QA; un alto escape indica fallas en el proceso de pruebas.
- **Riesgo operativo**: relación entre número de fallas críticas y total de transacciones; se puede calcular semanalmente.
- **Cobertura de pruebas**: porcentaje de código o funcionalidades cubiertas por pruebas automatizadas; el objetivo inicial es $\geq$ 70 % para componentes críticos.
- **Satisfacción del usuario**: feedback de comercios piloto (encuestas de NPS); se incorpora como métrica cualitativa.
- **Performance efficiency**: basándose en ISO 25010, se mide el tiempo de respuesta y utilización de recursos [14]. Se definen SLOs para tiempos de carga y procesamiento de ventas.

Indicadores de seguridad y mantenimiento

- **Incidentes de seguridad**: número y severidad de vulnerabilidades detectadas (p.ej., inyecciones de datos, accesos no autorizados). La subcaracterística **integridad** del modelo ISO 25010 enfatiza proteger el sistema contra modificaciones no autorizadas [15].
- **Frecuencia de actualizaciones**: número de releases de mantenimiento por trimestre; ayuda a mantener la mantenibilidad elevada.
- **Tiempo de actualización**: tiempo necesario para desplegar una nueva versión en todos los comercios; debe ser $\leq$ 2 horas durante la noche.

---

10. Contrato de liberación (QA Release Contract)

Antes de desplegar una versión a producción (comercios piloto o clientes), el equipo debe llenar y firmar un documento que certifique que se han cumplido todos los criterios. A continuación se presenta una plantilla mínima, adaptable a PDF o Markdown.

```
# Contrato de liberación – Nordia POS vX.X.X

## Resumen
- **Versión**: vX.X.X
- **Fecha prevista de despliegue**: AAAA-MM-DD
- **Objetivo principal**: (nueva funcionalidad, corrección de bugs, etc.)

## Checklist de calidad

| Elemento | Cumplido (Sí/No) | Evidencia |
|---|---|---|
| Historias de usuario completadas y revisadas | | |
| Revisión de código aprobada | | |
| Pruebas unitarias (≥ 80 % cobertura) | | |
| Análisis estático (Analyzer) sin hallazgos críticos | | |
| Pruebas dinámicas (Inspector) completadas | | |
| Escenarios de Pilot ejecutados y aprobados | | |
| Smoke tests manuales pasados al 100 % | | |
| Defectos críticos o urgentes pendientes | | |
| Documentación actualizada (plantillas de bug y test) | | |
| Métricas SLIs y SLOs dentro de objetivos | | |
| Firmas de responsables | | |
```

```
## Observaciones
- Incluir cualquier riesgo conocido, workaround temporal o deuda técnica
aceptada.

## Firmas
- **Director Técnico QA**: _____
- **Auditor Externo QA**: _____
- **Founder & QA Lead**: _____
- **Fecha de aprobación**: _____
```

La liberación **no se considerará válida** hasta que todas las casillas estén marcadas como “Sí” y las firmas estén presentes. Cualquier excepción debe documentarse en “Observaciones” y ser aprobada por el Founder.

---

11. Integración con la arquitectura de 4 motores

El estándar se integra con la arquitectura propuesta por Claude y refinada por ChatGPT. Cada motor añade valor en una capa diferente:

1. **Analyzer**: motor de análisis estático y contextual. Utiliza la API de IA para revisar HTML/JS del POS, generar prompts de evaluación, sanitizar entradas y sugerir casos de prueba. Este estándar le provee listas de escenarios rápidos y plantillas de bug/test para uniformar la salida.
2. **Inspector**: motor de análisis dinámico. Emplea Playwright en un backend Node.js ligero para ejecutar los smoke tests y escenarios de regresión. Exporta datos de tiempos de carga, errores de consola y captura de pantalla. Las métricas de performance (SLOs) se verifican aquí.
3. **Pilot**: motor de ejecución asistida por IA. Genera scripts Playwright a partir de descripciones en lenguaje natural y los ejecuta en sandbox. Si detecta un flujo no contemplado, pausa y solicita confirmación. Esta capa nutre de información al Brain y alimenta la base de conocimientos de Supabase.
4. **Brain** (futuro v4): motor de aprendizaje y recomendación. Analiza historiales de bugs, resultados de pruebas y métricas para predecir áreas de riesgo y sugerir priorizaciones. Este estándar define los datos que el Brain necesita (bug reports, casos de prueba, métricas SLIs/MTTR/MTBF) y cómo etiquetarlos.

Esta integración asegura que la calidad no sea un “afterthought” sino un proceso continuo de retroalimentación y mejora.

---

12. Roadmap sugerido (modo ambicioso)

Para poner en práctica este estándar y alcanzar una calidad de nivel enterprise con recursos actuales, se propone un roadmap en tres fases. Estas fases se integran con las propuestas de Claude pero se adaptan al calendario real (diciembre 2025 – marzo 2026).

Fase A – Orden y Estándares (primeras 2 semanas)

- Finalizar este documento y publicarlo en el repositorio de Nordia (QA). Crear versiones en español e inglés si es necesario.

- Configurar repositorio de pruebas con la plantilla JSON de bug y de caso de prueba; iniciar base de datos de Supabase para almacenar historial.
- Implementar la **Lista de Smoke Tests** y asegurarse de que todos pasen en la versión actual.
- Capacitar al equipo en la matriz de severidad y en la plantilla de bug.
- Definir SLOs iniciales para tiempos de respuesta y disponibilidad.

Fase B – QA Agent v3 con Inspector (mes siguiente)

- Desarrollar el backend Node.js del Inspector con Playwright y generar scripts para los 10 smoke tests.
- Integrar el Analyzer con el inspector para que, al detectar cambios en la base de código, genere automáticamente casos de prueba y los ejecute.
- Configurar métricas (SLIs) en Supabase o en otro servicio de monitoreo para MTTR, MTBF, etc.
- Probar el flujo completo (Analyzer → Inspector → manual) en los comercios piloto.

Fase C – Pilot MVP y Brain (enero – marzo 2026)

- Implementar la capa Pilot: generación de scripts a partir de prompts en lenguaje natural; integración con los escenarios rápidos.
- Definir e implementar la base de datos de conocimiento (Brain) en Supabase, indexando bugs, casos de prueba y resultados.
- Añadir capacidades de aprendizaje: priorización automática de pruebas en función de métricas e historial de fallos; sugerencia de nuevos SLOs.
- Revisar el estándar y actualizarlo a v2.0 con lecciones aprendidas.

Conclusión

El **Nordia QA Standard v1.0 – ChatGPT Zenith** es un documento vivo que establece las bases para una cultura de calidad en Nordia POS. Combina teoría (ISO 25010, SRE) y práctica (plantillas, checklists, métricas) para guiar a un equipo reducido pero ambicioso. Su objetivo no es burocratizar sino **acelerar**: permitir que cada miembro del equipo sepa qué esperar, cómo medir y cuándo aprobar. En palabras simples, que un carnicero en Corrientes pueda vender sin preocuparse por fallas, porque detrás existe un proceso riguroso que vela por la **calidad real** del producto.

---

1 2 15 ISO 25010
https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

3 4 5 6 Bug Severity Guide: Understanding the Criticality of Software Defects - DEV Community
https://dev.to/morrismoses149/bug-severity-guide-understanding-the-criticality-of-software-defects-2e64

7 SLOs, SLIs, and SLAs: Meanings & Differences | New Relic
https://newrelic.com/blog/best-practices/what-are-slos-slis-slas

8 9 10 11 12 13 What's the difference between MTTR, MTBF, MTTD, and MTTF | LogicMonitor
https://www.logicmonitor.com/blog/whats-the-difference-between-mttr-mttd-mttf-and-mtbf

14 ISO 25010
https://iso25000.com/en/iso-25000-standards/iso-25010