Prueba Técnica: Ingeniero de Calidad Funcional

Daniel Oswaldo Contreras Torres

Fase 1 – Análisis y Estrategia de Optimización

1.1 Priorización de funcionalidades para automatización

Basado en el riesgo para el negocio y volatilidad de los cambios, propongo las funcionalidades así (de mayor a menor prioridad para automatizar):

- Checkout y Pago

  Riesgo: Es el corazón del negocio. Si el flujo de pago falla, se afecta directamente la parte financiera de la empresa y los clientes.

  Volatilidad: Alta. Involucra reglas de negocio, distintos medios de pago e integración con terceros (pasarelas de pago, bancos, etc.).

  Justificación: Cualquier error afecta directamente ingresos, reputación y confianza del cliente. Es el primer candidato para automatización intensiva (UI + API) y validaciones regresivas frecuentes.

- Búsqueda y Filtros

  Riesgo: Impacta la capacidad del usuario para encontrar productos y de esta forma poder generar flujos transaccionales complejos.

  Volatilidad: Media, pero con una gran cantidad de datos, reglas de filtrado y dependencias de API.

  Justificación: Una búsqueda o filtrado defectuoso podría reducir ventas aunque el checkout funcione bien. Además, suele tener gran cantidad de casos de prueba (combinaciones de filtros, ordenamientos, paginación), lo que hace que la automatización genere un ahorro importante frente a la ejecución manual.

- Login y Perfil

  Riesgo: No puedan ingresar al aplicativo y gestionar sus perfiles

  Volatilidad: Baja, con pocas fallas reportadas y no genera un impacto a nivel monetario.

  Justificación: Aunque es crítico para el acceso, los cambios suelen ser menos frecuentes y los flujos son más simples.

- Panel de Administración

  Riesgo: no se pueden generar configuraciones avanzadas del sistema o usuarios con bloqueos.

  Volatilidad: Baja, pero con alta complejidad de UI.

  Justificación: Tiene menos impacto directo en el ingreso diario del e-commerce. Los cambios en estos módulos son mínimos dado que los cambios en usuarios y roles se realizan pocas veces.

1.2 ROI – En qué funcionalidad invertiría el 60% del tiempo (4 semanas)

Dedicaría aproximadamente el 60% del tiempo de automatización a la funcionalidad de Checkout y Pago.

Justificación:

- Es la etapa donde se concreta la venta y flujos monetarios.
- Un fallo en este flujo puede significar pérdida directa de ingresos, abandono masivo de carritos y quejas de los clientes.
- Combina muchos escenarios: distintos medios de pago, diferentes tipos de productos, cupones, costos de envío, impuestos, errores de la pasarela, respuestas lentas o caídas de terceros.
- Estos escenarios suelen ocupar una parte importante de las 12 horas de regresión manual.
- Automatizando la mayoría de los flujos de Checkout y Pago (idealmente en API y algunos en UI de punta a punta), se pueden realizar ejecuciones en horarios no laborales liberando recursos importantes de los ambientes.
- Esto permite ejecutar regresiones completas con mayor frecuencia (por ejemplo, en cada release o incluso por cada merge importante) sin depender de disponibilidad manual del equipo.
- Detección temprana de defectos críticos
- Al estar automatizados los flujos clave de pago, los errores graves (por ejemplo, cálculo incorrecto del total, rechazo inesperado, no generación de orden) se detectan de forma temprana, antes de llegar a producción.

2. Estrategia de Optimización (Automatizar para el Ahorro)

2.1 Diseño de “Pirámide Invertida” orientada a API para Búsqueda/Filtros y Checkout

La propuesta es realizar la mayor parte de las pruebas de regresión de estas dos funcionalidades a la capa de API, y mantener en la capa de UI solo smoke test o ruta critica

Estrategia para Búsqueda y Filtros:

- Identificar los endpoints de búsqueda que usa la aplicación (por ejemplo, /search, /products, /filters).
- Automatizar en API, con una librería como Serenity Rest en Java, los siguientes tipos de pruebas:
    - Búsqueda por texto (términos simples y combinados).
    - Validación de filtros (categoría, rango de precios, etc.).

Mantener en UI solo unos pocos escenarios representativos, por ejemplo:

- Búsqueda básica desde la barra principal.
- Aplicación de 1–2 filtros combinados y verificación visual de resultados.

Estrategia para Checkout y Pago:

- Identificar los servicios que soportan el checkout, por ejemplo:

- Creación y actualización del carrito (/cart, /cart/items).
- Cálculo de totales (/checkout/summary).
- Procesamiento de pago (/payment, /orders).

Automatizar en API:

- Cálculo correcto de subtotales, impuestos, descuentos y costos de envío.
- Diferentes métodos de pago (tarjeta, transferencia, contraentrega, etc.).
- Validación de estados de la transacción (aprobado, rechazado, pendiente).
- Manejo de errores de terceros (timeout, respuesta inválida, rechazos de la pasarela).

Dejar solo 1 o 2 flujos UI end-to-end:

- Cliente inicia sesión, agrega productos, realiza checkout y llega a la página de confirmación.
- Con esto se valida que la integración entre capas (UI + API) sigue siendo correcta.
- Por qué esta estrategia genera un ahorro considerable:

Tiempo de ejecución:

> Las pruebas de API son mucho más rápidas que las pruebas UI (no hay renderizado de navegador ni interacción visual). Esto permite ejecutar cientos de casos en el tiempo que antes se ejecutaban unos pocos manualmente.

Menor inestabilidad:

> Al no depender del navegador, del DOM ni de animaciones, se reducen fallas por elementos no encontrados, tiempos de carga, cambios menores de UI, etc.

2.2 Manejo de Datos para Checkout mediante API

Para reducir la dependencia de datos manuales se pueden crear escenarios que realicen la preparación y limpieza de datos antes de cada caso de prueba de Checkout, apoyado en servicios de API. Por ejemplo:

- Vaciar carritos de compra.
- Verificar inventarios de productos.
- Crear cupones de descuento o reglas de envío, etc.

Otro enfoque podría ser la consulta de base de datos en el cual se extraen datos de forma dinámica que cumplan las condiciones de datos requeridos para los escenarios de pruebas.