Plan Estratégico de QA: Modernización y Optimización de Pruebas de Regresión

Objetivo KPI: Reducir el Tiempo de Mercado (TTM) de la suite de regresión de 12 horas a < 2 horas.
Contexto: E-commerce con alta transaccionalidad y arquitectura híbrida.

Resumen Ejecutivo

Este documento detalla la estrategia técnica y metodológica para optimizar el ciclo de pruebas de regresión. Actualmente, el tiempo de ejecución (12 horas) representa un cuello de botella para el despliegue continuo. La solución propuesta combina una **metodología de selección basada en riesgos** (para reducir el alcance innecesario) con una **modernización de arquitectura de pruebas** (paralelismo y API-First) para acelerar la ejecución técnica.

1. Metodología: Regresión Basada en Riesgos (Risk-Based Testing)

Para alcanzar la meta de < 2 horas, abandonamos la ejecución ciega del 100% de la suite en favor de una ejecución inteligente.

1.1. Sistema de Puntuación (Scoring)

Cada funcionalidad se evalúa antes del ciclo de regresión bajo tres criterios ponderados:

1. **Volatilidad (V):** Frecuencia de cambios en el código reciente.
   - *1: Baja (Código Legacy/Estable)*
   - *3: Alta (Código Nuevo/Refactor)*
2. **Probabilidad de Fallo (P):** Basada en la deuda técnica y el historial de bugs.
   - *1: Baja*
   - *3: Alta*
3. **Impacto en Negocio (I):** Severidad si la funcionalidad falla en producción.
   - *1: Menor (Cosmético)*
   - *4: Crítico (Bloqueante/Pérdida de Ingresos)*

Fórmula de Prioridad: Score = Volatilidad + Probabilidad + Impacto

1.2. Matriz de Decisión y Ejecución

Establecemos un umbral de corte en **6 puntos** para la regresión automatizada completa.

Funcionalidad	Volatilidad (1-3)	Prob. Fallo (1-3)	Impacto (1-4)	Score Total	Estrategia de Ejecución
Checkout y Pagos	3	3	4	10	Full Regresión (Crítica)
Búsqueda y Filtros	2	2	3	7	Full Regresión (API Priority)
Login y Perfil	1	1	4	6	Smoke Test (Happy Path)
Panel Admin	1	1	1	3	Excluir (Manual / Visual)

2. Análisis de Valor y Priorización por Áreas

Basado en la matriz anterior, definimos la profundidad de las pruebas para maximizar el ROI.

2.1. Checkout y Pasarelas de Pago (Prioridad Alta - Score 10)

- **Justificación:** Es el "corazón" financiero. Alta volatilidad por integraciones de terceros y alto riesgo.
- **Enfoque:** Automatización exhaustiva.
- **Acción:** Cubrir flujos positivos, negativos, rechazos de tarjeta y cálculos de impuestos.

2.2. Búsqueda y Navegación (Prioridad Media - Score 7)

- **Justificación:** Puerta de entrada a la conversión. Si el usuario no encuentra, no compra.
- **Enfoque:** Pruebas de Integración.
- **Acción:** Validar la lógica de filtrado y ordenamiento masivamente vía API, dejando una capa mínima de UI.

2.3. Login y Gestión de Usuarios (Prioridad Baja - Score 6)

- **Justificación:** Funcionalidad crítica pero de muy baja volatilidad (rara vez cambia).
- **Enfoque:** Mantenimiento Mínimo.

- **Acción:** Scripts robustos de "Smoke Test". Si el login funciona, asumimos estabilidad en el resto del perfil.

2.4. Panel de Administración (Prioridad Mínima - Score 3)

- **Justificación:** Uso interno, bajo impacto en ingresos directos, alta complejidad de UI.
- **Enfoque:** No automatización funcional.
- **Acción:** Delegar a pruebas exploratorias manuales o pruebas de regresión visual (snapshots).

3. Arquitectura Técnica de la Solución

Cómo logramos técnicamente la reducción de tiempos (de 12h a 2h).

3.1. Pirámide Invertida (Estrategia API-First)

Movemos la validación de reglas de negocio de la capa lenta (UI) a la capa rápida (API).

- **Búsqueda y Filtros:**
  - *Actual:* UI Test tarda 2 mins en filtrar por "Color Rojo + Talla M".
  - *Nuevo:* API Test tarda 200ms en validar el JSON de respuesta. Cubrimos 100 combinaciones en API en el tiempo de 1 test de UI.
- **Checkout:**
  - Validación de impuestos, descuentos y creación de órdenes directamente contra los microservicios.

3.2. Infiraestructura: Paralelismo y Sharding

La ejecución secuencial es el enemigo de la velocidad.

- **Sharding (Fragmentación):** Dividiremos la suite de pruebas en múltiples "shards" (fragmentos).
- **Ejecución Paralela:** Implementación de 4 a 6 agentes (workers) en el pipeline de CI/CD.
  - *Cálculo:* 1000 tests / 5 agentes = 200 tests por agente.
  - *Impacto:* Reducción directa del tiempo total de ejecución en un factor de 4x a 5x.

3.3. Gestión de Datos (Data Management & State Injection)

Eliminación de tiempos muertos por preparación de datos y maximización del valor para otras áreas.

- **Data on Demand (Atomicidad):**
  - Cada test crea sus propios datos (Usuario, Producto) vía API antes de comenzar.
- **Inyección de Estado (State Injection):**
  - Para probar el Checkout, **no navegamos** desde el Home.
  - El script llama a la API de Login, obtiene el Token/Cookie, lo inyecta en el navegador y abre directamente /checkout/payment.
  - *Ahorro:* Se eliminan 30-45 segundos de navegación innecesaria por cada test.

- **Beneficio Extendido: Generación Masiva para Performance y Dev:**
  - **Pruebas de Carga:** Los scripts de generación de datos vía API se reutilizan para inyectar miles de registros (ej: 50,000 órdenes) en minutos, facilitando pruebas de estrés (JMeter/K6) sin esfuerzo adicional.
  - **Soporte a Desarrollo:** Los desarrolladores pueden utilizar estos scripts para poblar sus entornos locales con "datos frescos" y válidos instantáneamente, mejorando la calidad de sus pruebas unitarias y reduciendo la dependencia de dumps de base de datos obsoletos.

4. Hoja de Ruta (Roadmap) - Plan de 4 Semanas

Semana	Fase	Actividades Clave
Semana 1	**Infraestructura y Clasificación**	1. Implementar "Tagging" en los tests según la Matriz de Riesgos. 2. Configurar Sharding (Paralelismo) en el CI/CD (Github Actions/Jenkins).
Semana 2	**Shift-Left (API Migration)**	1. Migrar el 80% de los casos de prueba de "Búsqueda y Filtros" de UI a API (RestAssured/Playwright API). 2. Reducir la suite de UI de búsqueda a solo "Happy Paths".
Semana 3	**Optimización Checkout**	1. Implementar patrón de "Inyección de Estado" para saltar Login y Home en tests de Checkout. 2. Implementar scripts de generación de datos API reutilizables para QA y Dev.
Semana 4	**Estabilización y Corte**	Activar la ejecución condicional basada en matriz en el pipeline. 1. 2. Medición final de tiempos y ajuste de recursos (workers).

5. Conclusión y ROI Esperado

Con la implementación de esta estrategia, transformamos el proceso de QA de un cuello de botella a un habilitador de velocidad y calidad transversal.

- **Reducción de Tiempo:** 83% (De 12 horas a ~2 horas).
- **Cobertura:** Más inteligente, enfocada 100% en donde está el riesgo financiero.
- **Valor Agregado:** Capacidad de generar datos sintéticos masivos para pruebas de

performance y soporte al equipo de desarrollo.

Anexo A: Glosario de Términos y Abreviaciones

Para facilitar la lectura de este documento técnico, se definen los siguientes términos clave:

Conceptos de Negocio y Métricas

- **KPI (Key Performance Indicator):** Indicador Clave de Desempeño. Métrica utilizada para cuantificar el éxito de un objetivo específico (en este caso, el tiempo de ejecución de pruebas).
- **TTM (Time to Market):** Tiempo de Comercialización. El tiempo que transcurre desde que se concibe una funcionalidad hasta que está disponible para los clientes.
- **ROI (Return on Investment):** Retorno de la Inversión. Medida de rentabilidad que evalúa la eficiencia de una inversión (el esfuerzo de automatizar) comparado con el ahorro generado.

Estrategias de Testing

- **Risk-Based Testing:** Pruebas Basadas en Riesgos. Metodología que prioriza la ejecución de casos de prueba basándose en la probabilidad de falla y el impacto en el negocio, en lugar de probar todo por igual.
- **Shift-Left:** Estrategia de "mover a la izquierda" las pruebas en el ciclo de desarrollo. Significa probar más temprano (API, Unitarias) en lugar de esperar al final (UI).
- **API-First Testing:** Enfoque que prioriza la validación de la lógica de negocio en la capa de servicios (backend) antes que en la interfaz gráfica, por ser más rápida y estable.
- **Smoke Test:** Prueba de Humo. Un conjunto mínimo de pruebas básicas diseñadas para verificar que las funciones críticas del sistema (ej. Login, Carga de Home) funcionan, sin entrar en detalles profundos.
- **Happy Path:** "Camino Feliz". El flujo ideal de un usuario donde no ocurren errores ni

excepciones (ej. un usuario compra un producto con saldo suficiente y sin errores de red).

Infraestructura y Tecnología

- **CI/CD (Continuous Integration / Continuous Deployment):** Práctica de desarrollo de software donde los cambios en el código se construyen, prueban y despliegan de forma automática y frecuente.
- **Sharding (Fragmentación):** Técnica de dividir una gran suite de pruebas en partes más pequeñas ("shards") para ejecutarlas simultáneamente en diferentes máquinas.
- **Flaky Tests:** Tests "Intermitentes" o inestables. Pruebas que a veces pasan y a veces fallan sin que haya cambios en el código, generalmente debido a problemas de datos, red o tiempos de espera.
- **State Injection:** Inyección de Estado. Técnica avanzada donde se manipula el navegador (cookies, local storage) para poner la aplicación en un estado específico (ej. "usuario logueado") instantáneamente, saltándose los pasos manuales de la UI.
- **Data on Demand:** Datos bajo demanda. Creación programática de datos de prueba justo en el momento en que se necesitan, garantizando que sean válidos y únicos para ese test.