# Backlog Funcional del Proyecto CarMatch (Expansión)

A continuación se presenta el **backlog funcional** expandido del proyecto CarMatch, estructurado en 5 épicas. Cada épica contiene 5 historias de usuario identificadas con un código único (HU-XXX). Las historias se redactan siguiendo el formato *“Como [rol], quiero [necesidad] para [beneficio]”* e incluyen criterios de aceptación en estilo BDD (Dado/Cuando/Entonces) y la prioridad asignada según MoSCoW. Se abordan aspectos detallados de interacción: validaciones, gestión de errores, estados vacíos, experiencia de usuario (UX) y flujos alternativos. Las historias están ordenadas progresivamente dentro de cada épica (de funcionalidades básicas a avanzadas), cubriendo exhaustivamente los requerimientos funcionales identificados del proyecto. Al final, se proporciona la tabla de trazabilidad actualizada que relaciona cada requerimiento funcional con sus historias de usuario, así como los criterios de DoR, DoD, requisitos no funcionales y la priorización MoSCoW de todo el backlog.

## Épica 1: Integración de Fuentes de Datos y Web Scraping Automático

*Descripción:* Esta épica abarca la **recolección automatizada de datos** de repuestos y servicios automotrices desde múltiples fuentes en línea mediante web scraping. Incluye la configuración y gestión de las fuentes de datos, la ejecución periódica de los scrapers, el manejo de errores en estos procesos y la garantía de calidad de los datos (unificación y limpieza) antes de almacenarlos. El objetivo es mantener una base de datos central **actualizada y confiable** con información consolidada de diversos portales automotrices sin intervención manual.

### HU-001: Como administrador del sistema, quiero que el proceso de scraping se ejecute automáticamente en intervalos regulares sobre todas las fuentes definidas, para mantener la información de repuestos siempre actualizada en la plataforma.

* **Criterios de Aceptación:**
* *Dado* que existe un conjunto de fuentes de datos configuradas, *cuando* llega la hora programada (ej. diariamente a medianoche) para ejecutar la tarea de scraping, *entonces* el sistema inicia automáticamente la recolección de datos desde cada fuente activa y actualiza la base de datos con los nuevos repuestos o cambios detectados.
* *Dado* que un portal fuente está temporalmente inaccesible o presenta fallas durante la ejecución, *cuando* ocurre el scraping automático, *entonces* el sistema continúa con las demás fuentes sin interrumpir el proceso general, registrando en logs el error de la fuente fallida para revisión posterior.
* *Dado* que una fuente ya fue procesada recientemente, *cuando* se ejecuta nuevamente el scraping automático, *entonces* el sistema solo extrae e incorpora los nuevos datos o cambios desde la última ejecución (evitando duplicados) y omite datos sin cambios, optimizando la actualización.
* **Prioridad:** Must (Obligatorio)

### HU-002: Como administrador, quiero **agregar, editar o remover fuentes de datos** para el scraping mediante la interfaz de administración, para ampliar o depurar la cobertura de la plataforma fácilmente sin cambios de código.

* **Criterios de Aceptación:**
* *Dado* que dispongo de los datos de una nueva fuente (URL base, selectores o API, credenciales si aplica), *cuando* ingreso esos datos en el formulario “Añadir Fuente” del panel admin y confirmo, *entonces* la nueva fuente se registra en el sistema, queda listada como **activa** y será incluida en las próximas ejecuciones automáticas de scraping.
* *Dado* que una fuente existente cambia su estructura o requiere ajustes, *cuando* el administrador modifica la configuración de la fuente (por ejemplo, actualiza la URL o los selectores CSS/XPath de extracción) y guarda los cambios, *entonces* el sistema utiliza la configuración actualizada en la siguiente ejecución de scraping, reflejando los ajustes para obtener datos correctamente.
* *Dado* que una fuente ya no es relevante o confiable, *cuando* el administrador la desactiva o elimina del panel, *entonces* dicha fuente se excluye de las tareas de scraping futuras y sus datos pueden marcarse como obsoletos (sin eliminar históricos a menos que se indique) para no seguir mostrándolos a los usuarios.
* *Dado* un escenario donde se agrega una fuente duplicada o ya existente, *cuando* el administrador intenta guardarla, *entonces* el sistema debe impedir la duplicación mostrando un mensaje de error claro (e.j., “La fuente ya está registrada”) y no permitir fuentes redundantes.
* **Prioridad:** Should (Debería)

### HU-003: Como administrador, quiero que el sistema registre **errores y eventos de scraping** (por ejemplo, fallos de conexión, cambios en estructura de página) y me notifique de los problemas críticos, para poder detectar y solucionar incidencias rápidamente y asegurar la continuidad de la recolección de datos.

* **Criterios de Aceptación:**
* *Dado* que ocurre un error durante la ejecución de un scraper (por ejemplo, un portal bloquea el acceso o cambia su HTML), *cuando* finaliza el proceso de scraping, *entonces* el sistema almacena un registro detallado del evento (fuente afectada, timestamp, tipo de error, detalle técnico) en los **logs** y marca esa ejecución como fallida en el panel de administración.
* *Dado* que un error crítico impide obtener datos de una o más fuentes, *cuando* el sistema detecta este escenario (por ejemplo, falla en todas las fuentes o caída del servicio de scraping), *entonces* se envía una **alerta** al administrador (vía email o notificación en el panel) indicando la situación para una intervención oportuna.
* *Dado* que el administrador accede al panel de administración, *cuando* consulta la sección de estado de scraping, *entonces* puede ver un resumen de las últimas ejecuciones (éxito o fallo por fuente, tiempo de ejecución, número de elementos nuevos actualizados) junto con indicadores visuales (por ejemplo, íconos de advertencia en caso de errores) que facilitan identificar problemas.
* *Cuando* se resuelve un problema en una fuente (e.g., se actualiza la configuración tras un cambio de estructura), *entonces* en la siguiente ejecución exitosa el sistema debe reflejar el retorno a estado **OK** en los logs/alertas, de modo que el administrador vea que la fuente vuelve a operar normalmente.
* **Prioridad:** Must (Obligatorio)

### HU-004: Como **usuario de la plataforma**, quiero que los resultados de repuestos mostrados estén **unificados y sin duplicados** aunque provengan de distintas fuentes, para evitar confusión al comparar y asegurar que no veo el mismo producto repetido varias veces.

* **Criterios de Aceptación:**
* *Dado* que un mismo repuesto (mismo identificador único, código o descripción exacta) aparece en múltiples fuentes, *cuando* el proceso de integración almacena los datos, *entonces* el sistema **detecta entradas duplicadas** y conserva solo una ficha consolidada en la base de datos (pudiendo agregar referencia a las múltiples fuentes/orígenes si se desea), de manera que en la interfaz de búsqueda ese repuesto se muestre una sola vez al usuario.
* *Dado* que existan pequeñas variaciones entre datos duplicados (por ejemplo, precios ligeramente distintos, formato de moneda, nombres con mayúsculas/minúsculas), *cuando* se unifica la información, *entonces* el sistema normaliza los campos relevantes – por ejemplo, convirtiendo todos los precios a un formato único (CLP) y homogeneizando nombres de atributos – antes de guardar, asegurando consistencia en la comparación.
* *Cuando* el usuario realiza una búsqueda, *entonces* los resultados desplegados **no incluyen entradas duplicadas**; en caso de que un mismo producto tenga múltiples ofertas (varios proveedores), el sistema puede mostrarlo como una única entrada agregada (ej.: “Disponible en 3 tiendas”) o mostrar solo la oferta más relevante (ej.: la de menor precio) según criterios definidos, en lugar de resultados repetidos.
* *Dado* que por error llegasen a mostrarse duplicados en la interfaz, *cuando* el usuario note elementos idénticos repetidos, *entonces* esto se considera un defecto y deberá corregirse en la lógica de integración, garantizando en futuras iteraciones la **deduplicación** completa de la información presentada.
* **Prioridad:** Should (Debería)

### HU-005: Como administrador, quiero configurar la **frecuencia de ejecución** del scraping para cada fuente (por ejemplo, diariamente, semanalmente o en intervalos específicos) e incluso ejecutar una recolección manual bajo demanda, para mantener los datos actualizados optimizando el uso de recursos y evitar sobrecargar los sitios fuente.

* **Criterios de Aceptación:**
* *Dado* el listado de fuentes en el panel de administración, *cuando* el administrador establece para una fuente una periodicidad (por ejemplo, cada 6 horas, una vez al día a cierta hora, lunes a viernes, etc.) y guarda la configuración, *entonces* el sistema programa las tareas de scraping conforme a esos intervalos para esa fuente, ajustando el calendario general de ejecuciones.
* *Dado* que una fuente específica requiere datos en tiempo real o se actualiza con poca frecuencia, *cuando* el administrador configura una frecuencia más alta (ej. cada hora) o más baja (ej. semanal) respectivamente, *entonces* el sistema respeta esa frecuencia para dicha fuente sin afectar las demás, permitiendo granularidad en la actualización según la naturaleza de cada origen de datos.
* *Dado* un escenario donde surge una necesidad urgente de actualizar los datos (por ejemplo, se detectó un error corregido o se lanzó una promoción en una tienda), *cuando* el administrador pulsa un botón de “Ejecutar scraping ahora” para una fuente o para todas en el panel, *entonces* el sistema inicia inmediatamente una tarea de scraping fuera del cronograma regular, actualizando la información al instante y registrando la ejecución manual en los logs.
* *Cuando* el administrador modifica o lanza una frecuencia de scraping, *entonces* la interfaz confirma la acción (por ejemplo, mostrando “Frecuencia actualizada” o “Scraping manual iniciado…”) y refleja el próximo horario programado de ejecución en la vista de cada fuente, brindando visibilidad sobre **cuándo** será la siguiente actualización.
* **Prioridad:** Could (Podría)

## Épica 2: Backend y API REST para Gestión de Datos

*Descripción:* Esta épica se centra en el desarrollo del **backend** del sistema utilizando Django (u otro framework adecuado) para exponer una **API REST** que permita acceder a los datos de repuestos almacenados. Incluye el diseño del modelo de datos en PostgreSQL (en la nube, GCP) y la implementación de diversos endpoints (búsqueda, detalle, comparación) que serán consumidos por el frontend. También abarca aspectos de seguridad en la API (autenticación/autorización, validación de entradas, protección contra ataques) y consideraciones de integridad y consistencia de la **persistencia** de datos. El objetivo es proveer una capa de servicios robusta y escalable que sirva de puente entre la base de datos y la interfaz de usuario.

### HU-006: Como desarrollador front-end, quiero un **endpoint REST de búsqueda** que permita consultar repuestos por palabra clave y filtros, para obtener desde el frontend los resultados relevantes almacenados en la base de datos mediante llamadas HTTP.

* **Criterios de Aceptación:**
* *Dado* que el cliente realiza una petición GET al endpoint de búsqueda (por ejemplo, /api/repuestos/?q=<texto>&categoria=<cat>&precio\_min=X&precio\_max=Y), *cuando* la petición incluye un término de búsqueda válido, *entonces* la API responde con código 200 y un **JSON** que contiene la lista de repuestos que coinciden con el texto y criterios proporcionados, incluyendo campos clave como nombre del repuesto, categoría, precio, tienda fuente, y un identificador único de cada ítem.
* *Dado* que se especifican parámetros de filtro (categoría, rango de precio, etc.), *cuando* se realiza la petición al endpoint, *entonces* la API aplica esos filtros en la consulta a la base de datos y retorna únicamente los resultados que satisfacen dichas condiciones (por ejemplo, precio entre X e Y, categoría = “Filtros de aceite”), excluyendo los que no correspondan.
* *Dado* que la búsqueda no encuentra ningún resultado, *cuando* se realiza la petición con esa palabra o filtros, *entonces* la API responde con éxito (200) pero con una lista vacía ([]) o un objeto indicando “sin resultados”, de modo que el frontend sepa que no hay coincidencias y pueda manejar ese estado (por ejemplo, mostrando un mensaje al usuario).
* *Dado* que la petición está malformada o falta un parámetro requerido, *cuando* el servidor recibe la solicitud, *entonces* la API devuelve un código de error 400 **Bad Request** con un mensaje descriptivo (ej.: “Parámetro ‘q’ es obligatorio”) para que el desarrollador front-end pueda depurar la llamada.
* *Cuando* se realizan múltiples búsquedas concurrentes o la base de datos contiene un volumen grande de datos, *entonces* el endpoint debe responder en tiempos aceptables (p.ej., <2 segundos para consultas típicas bajo carga normal), utilizando paginación de resultados (e.g., 20 resultados por página con enlaces o parámetros para página siguiente) y consultas optimizadas con índices para mantener el desempeño.
* **Prioridad:** Must (Obligatorio)

### HU-007: Como desarrollador front-end, quiero un **endpoint REST para obtener el detalle completo** de un repuesto específico, de forma que al seleccionar un item en la interfaz pueda consultar información detallada (descripción, características, historial, etc.) desde el backend.

* **Criterios de Aceptación:**
* *Dado* un identificador único de repuesto (ID) válido, *cuando* se realiza una petición GET al endpoint de detalle (por ejemplo, /api/repuestos/<id>/), *entonces* la API responde con 200 OK y retorna un objeto JSON con todos los detalles almacenados del repuesto correspondiente: nombre completo, descripción extensa, precio actual, moneda, categoría, proveedor/tienda de origen, URL de origen, imágenes (si se almacenan URLs), fecha de última actualización y cualquier otro atributo relevante (ej.: compatibilidad, marca, modelo del vehículo, etc. si aplica).
* *Dado* que el repuesto consultado tiene **datos históricos** registrados (por ejemplo, cambios de precio a lo largo del tiempo), *cuando* el cliente solicita este detalle, *entonces* la respuesta incluye también una estructura de datos para el historial (p. ej., una lista de puntos con fecha y precio) o un enlace a otro endpoint específico de historial (e.g., /api/repuestos/<id>/historial\_precios/), de modo que el frontend pueda graficar o mostrar la tendencia de precios en la UI.
* *Dado* que el ID proporcionado no existe en la base de datos (repuesto no encontrado o ya removido), *cuando* se realiza la petición, *entonces* la API devuelve un 404 **Not Found** con un mensaje (“Repuesto no encontrado”) sin exponer detalles internos, para indicar claramente el error.
* *Cuando* se consulta un detalle, *entonces* el sistema garantiza que la información provista corresponde a la versión más reciente en la BD (incluyendo actualizaciones provenientes del último scraping ejecutado) y que los datos relacionados (ej. referencias foráneas a tablas de categoría o tienda) están resueltos completamente (p. ej., entregar el nombre legible de la categoría en lugar de solo un código).
* **Prioridad:** Should (Debería)

### HU-008: Como desarrollador front-end, quiero un **endpoint REST de comparación** que reciba múltiples IDs de repuestos y retorne sus datos en una sola respuesta, para simplificar la obtención simultánea de detalles al comparar varios productos en la interfaz.

* **Criterios de Aceptación:**
* *Dado* una lista de identificadores de repuesto (por ejemplo, ids=45,78,112), *cuando* se realiza una petición GET al endpoint de comparación (e.g., /api/comparacion/?ids=45,78,112), *entonces* la API responde con 200 OK y retorna un JSON que contiene un **arreglo de objetos**, cada uno con los detalles relevantes de cada ID solicitado (similar a la información de HU-007 pero resumida si se desea), para que el frontend pueda mostrarlos lado a lado en la tabla de comparación.
* *Dado* que uno o más IDs proporcionados no existen o están inactivos, *cuando* se hace la petición, *entonces* la API responde con un error 400 indicando cuáles IDs son inválidos o, alternativamente, retorna sólo los datos de los IDs válidos junto con un campo de advertencia sobre los IDs ignorados, asegurando que no falle todo el proceso por un elemento incorrecto.
* *Dado* que este endpoint podría ser llamado frecuentemente con combinaciones repetidas de IDs (por ejemplo, muchos usuarios comparando los mismos productos populares), *cuando* se reciben dichas peticiones, *entonces* el sistema debe aprovechar una **caché selectiva**: si recientemente se solicitó la misma combinación de IDs, se puede devolver la respuesta desde caché en lugar de consultar la base de datos nuevamente, disminuyendo la latencia y la carga en el servidor.
* *Cuando* se retorna la respuesta de comparación, *entonces* el formato de salida alinea los datos homogéneamente (por ejemplo, cada objeto contiene los mismos campos en el mismo orden) para facilitar al frontend iterar y renderizar comparativas; además, se limita razonablemente la cantidad de ítems comparables simultáneamente (p. ej., el endpoint podría soportar hasta 5 IDs para no sobrecargar la respuesta).
* **Prioridad:** Could (Podría)

### HU-009: Como administrador, quiero que el **API REST esté securizada** con mecanismos de autenticación y autorización apropiados, además de validaciones contra ataques comunes, para proteger los datos sensibles y evitar un uso malicioso de los servicios del backend.

* **Criterios de Aceptación:**
* *Dado* que ciertas operaciones del API son de administración o potencialmente sensibles (ej.: endpoints para gestionar fuentes, usuarios, o modificar datos), *cuando* un cliente realiza una petición a dichos endpoints sin proveer credenciales válidas (token o login), *entonces* la API debe responder con 401 **Unauthorized** o 403 **Forbidden** según corresponda, **denegando** el acceso y protegiendo esas funciones de usuarios no autenticados.
* *Dado* un administrador con credenciales válidas, *cuando* envía el **token de autenticación** o inicia sesión correctamente y realiza peticiones a los endpoints seguros, *entonces* la API le permite el acceso (200 OK en las respuestas) y ejecuta la operación solicitada (por ejemplo, agregar una fuente) con éxito, aplicando controles de autorización para que solo usuarios con rol adecuado puedan efectuar cambios críticos.
* *Cuando* se reciben datos de entrada a la API (parámetros, cuerpo JSON), *entonces* el backend realiza **validación** estricta de formatos, rangos y tipos (por ejemplo, rechazando inyecciones SQL, scripts maliciosos en texto, o formatos incorrectos) devolviendo errores 400 con mensajes claros si algo no pasa la validación. Asimismo, se implementan medidas contra ataques **CSRF** (si aplica en peticiones autenticadas vía sesión) y se sanitizan las salidas para prevenir **XSS** en caso de que datos maliciosos se almacenen y luego se envíen al frontend.
* *Cuando* un cliente realiza un volumen inusualmente alto de peticiones al API en un corto periodo, *entonces* el sistema aplica **rate limiting** (limitación de tasa) por IP o por token, bloqueando o retardando temporalmente respuestas con códigos 429 **Too Many Requests**, para evitar abuso de servicio o posibles ataques de denegación de servicio a pequeña escala.
* **Prioridad:** Must (Obligatorio)

### HU-010: Como desarrollador backend, quiero que los **datos recolectados se almacenen en PostgreSQL** de forma normalizada y consistente, para garantizar persistencia confiable y facilitar futuras consultas y escalabilidad del sistema.

* **Criterios de Aceptación:**
* *Dado* el diseño del sistema, *cuando* se modela la base de datos, *entonces* las entidades principales (repuestos, categorías, fuentes, etc.) están definidas en tablas con relaciones apropiadas (p. ej., cada repuesto referencia la fuente de origen y una categoría estándar) aplicando principios de **normalización** (evitando duplicación innecesaria de datos mediante FK, tablas de asociación para N:M si corresponde, etc.).
* *Cuando* el proceso de scraping inserta o actualiza la información en la base de datos, *entonces* se respetan las **restricciones** de integridad establecidas: por ejemplo, no se insertan repuestos sin categoría o fuente válida (claves foráneas obligatorias), no se duplican entradas con la misma clave única (según lógica de negocio, e.j., combinación de código de repuesto + fuente debe ser única), y se actualizan registros existentes en vez de duplicarlos si ya estaban almacenados.
* *Dado* que la aplicación está desplegada en la nube (GCP) con PostgreSQL, *cuando* se realiza la conexión y operaciones a la base de datos, *entonces* se asegura la **configuración adecuada** del entorno: la base de datos está accesible solo para la aplicación (firewall, SSL si es remoto), con credenciales seguras, y se manejan excepciones de conexión o consulta de forma que no provoquen caídas no controladas del sistema.
* *Cuando* aumenta el volumen de datos (p. ej., decenas de miles de repuestos almacenados), *entonces* la estructura de la base de datos y la API siguen funcionando eficientemente, pudiendo escalar vertical u horizontalmente. Se han creado **índices** en campos clave de búsqueda (como nombre de repuesto, categoría, rango de precios) para mantener tiempos de respuesta rápidos en las consultas definidas en los endpoints anteriores.
* *Dado* que la base de datos podría enfrentar operaciones concurrentes (lecturas de usuarios desde la API y escrituras desde el scraping simultáneamente), *cuando* ocurren esas situaciones, *entonces* el sistema mantiene la **consistencia** utilizando transacciones cuando corresponda (por ejemplo, al actualizar múltiples tablas relacionadas), evitando condiciones de carrera o lecturas sucias, y registrando cualquier conflicto o deadlock en logs para análisis.
* **Prioridad:** Must (Obligatorio)

## Épica 3: Interfaz de Usuario para Búsqueda y Filtrado (Frontend)

*Descripción:* Esta épica abarca la implementación del **frontend** de CarMatch en Next.js, enfocada en la funcionalidad de **búsqueda de repuestos** y refinamiento de resultados. Incluye la creación de una interfaz de usuario intuitiva y responsiva donde el usuario final puede ingresar términos de búsqueda, ver un listado consolidado de resultados relevantes y aplicar **filtros** (por categoría, rango de precio, etc.) u **ordenamientos** para acotar los resultados según sus preferencias. También se consideran las experiencias de usuario asociadas: manejo de situaciones de “no hay resultados”, mensajes de error en caso de problemas con la búsqueda, paginación de resultados si son numerosos, y en general asegurar que la interacción de búsqueda sea rápida, clara y centrada en la usabilidad.

### HU-011: Como usuario final, quiero **buscar repuestos o accesorios por palabra clave** mediante una barra de búsqueda en la plataforma, para obtener rápidamente un listado de las opciones disponibles que coinciden con lo que necesito.

* **Criterios de Aceptación:**
* *Dado* que el usuario ingresa un término de búsqueda válido (ej.: “amortiguador Toyota Yaris”) en la barra de búsqueda y confirma (presiona Enter o clic en buscar), *cuando* se envía la solicitud al servidor, *entonces* la interfaz muestra una **lista de resultados** que contiene los repuestos coincidentes: cada resultado en la lista exhibe al menos el nombre del repuesto, una breve descripción o especificación principal, el precio actual, y la tienda o fuente de donde proviene, todo en una tarjeta o fila fácilmente legible.
* *Cuando* los resultados de la búsqueda exceden cierta cantidad por página (ej.: 20 resultados), *entonces* la interfaz implementa **paginación** o carga progresiva: muestra un número limitado de resultados inicialmente y ofrece controles (páginas numeradas o botón “Cargar más”) para navegar a través de todos los resultados encontrados sin abrumar al usuario.
* *Cuando* se muestran los resultados, *entonces* están ordenados por un criterio por defecto lógico (por ejemplo, **relevancia** de la coincidencia con la palabra clave ingresada, o quizás por precio más bajo si no hay un ranking de relevancia claro), a menos que el usuario aplique posteriormente otro ordenamiento (ver HU-014).
* *Dado* que la interfaz es responsiva, *cuando* el usuario realiza una búsqueda desde un dispositivo móvil o ventana pequeña, *entonces* la lista de resultados se adapta (por ejemplo, las tarjetas se hacen verticales de ancho completo, con texto e imágenes ajustadas) para que sea fácil de scroll y lectura en pantallas pequeñas, manteniendo la funcionalidad completa.
* *Cuando* el usuario inicia la aplicación por primera vez o no ha ingresado nada aún, *entonces* se puede mostrar una **pantalla inicial** de búsqueda vacía con un mensaje de bienvenida o instrucciones (ej.: “Ingresa un nombre o código de repuesto para comenzar la búsqueda”) en lugar de una lista vacía, ofreciendo contexto sobre cómo usar la plataforma.
* **Prioridad:** Must (Obligatorio)

### HU-012: Como usuario, quiero **filtrar los resultados por categoría** (tipo de repuesto o servicio), para poder reducir la lista solo a aquellos ítems que sean relevantes a la clase de producto que me interesa.

* **Criterios de Aceptación:**
* *Dado* que la plataforma maneja categorías estandarizadas de productos (ej.: Motor, Frenos, Neumáticos, Accesorios, Servicios mecánicos, etc.), *cuando* el usuario despliega un menú o panel de filtros de categoría junto a los resultados, *entonces* se presenta un listado de categorías disponibles con casillas de verificación o botones.
* *Cuando* el usuario selecciona una o varias categorías (por ejemplo “Neumáticos” y “Llantas”) y aplica el filtro, *entonces* la interfaz actualiza la lista de resultados en tiempo real (vía consulta al API con parámetro de categoría) mostrando **solo los repuestos pertenecientes a esas categorías**, y ocultando/eliminando temporalmente los demás de la vista.
* *Cuando* uno o más filtros de categoría están activos, *entonces* la interfaz lo indica claramente (por ejemplo, mostrando las categorías seleccionadas en algún lugar o resaltadas) y ofrece una forma fácil de quitarlos (como una “X” para remover cada filtro o un botón “Limpiar filtros”), tras lo cual la lista vuelve a mostrar todos los resultados originales sin filtrar.
* *Dado* que puede haber categorías sin resultados para la búsqueda realizada, *cuando* el usuario selecciona una categoría que no tiene coincidencias en el conjunto actual (ej.: filtra “Baterías” pero ningún resultado de la búsqueda pertenece a esa categoría), *entonces* la lista aparecerá vacía y se mostrará un mensaje claro de “No hay resultados en esta categoría para tu búsqueda”, ayudando al usuario a entender que los datos existen pero no bajo ese filtro específico.
* *Dado* que el número de categorías podría ser alto, *cuando* se despliega el panel de categorías, *entonces* podría incorporarse una **búsqueda dentro de las categorías** (por ejemplo, un pequeño campo para filtrar la lista de categorías) o una estructura jerárquica si aplica, para que el usuario encuentre fácilmente la categoría deseada.
* **Prioridad:** Should (Debería)

### HU-013: Como usuario, quiero **filtrar los resultados por rango de precio**, para enfocarme solamente en las opciones que están dentro de mi presupuesto.

* **Criterios de Aceptación:**
* *Dado* que la lista de resultados inicial incluye repuestos con diversos precios, *cuando* el usuario ajusta un control de rango de precios (por ejemplo, un par de cajas de texto o un slider dual para mínimo y máximo) e ingresa un valor mínimo y/o máximo y luego aplica el filtro, *entonces* la interfaz actualiza la lista mostrando **solo aquellos resultados cuyo precio esté entre esos valores** (inclusive).
* *Cuando* el filtro de precio está activo, *entonces* en la interfaz se refleja (por ejemplo, mostrando “Precio: $10.000 - $50.000” como etiqueta activa) y cualquier resultado fuera de ese rango no se muestra. Si ningún resultado cae dentro del rango especificado, se presenta un mensaje de “No hay repuestos en ese rango de precio” similar al caso de categorías.
* *Dado* que el usuario puede ingresar valores inválidos (ej.: texto no numérico, rango invertido donde min > max, etc.), *cuando* se intenta aplicar el filtro con entradas incorrectas, *then* el sistema valida y no ejecuta la búsqueda; en su lugar, muestra un **mensaje de error** junto al control (por ejemplo, “Ingrese un número válido” o “El precio mínimo no puede ser mayor que el máximo”) indicando cómo corregirlo, sin recargar la página.
* *Cuando* se han aplicado simultáneamente filtros de categoría (HU-012) y de precio, *entonces* la plataforma combina ambos criterios (operación AND lógica): se muestran solo los resultados que pertenecen a las categorías seleccionadas **y** cuyo precio está dentro del rango dado, reflejando adecuadamente ambos filtrados en la interfaz y permitiendo quitarlos independientemente.
* *Dado* que los precios de los repuestos están en una moneda unificada (CLP, pesos chilenos) tras la normalización, *cuando* el usuario aplique el filtro, *entonces* no se presentan incongruencias de moneda. En caso de futuros alcances multi-moneda, el filtro se aplicaría sobre los precios convertidos a una divisa base para consistencia.
* **Prioridad:** Should (Debería)

### HU-014: Como usuario, quiero **ordenar los resultados** según distintos criterios (por ejemplo, por precio de menor a mayor, mayor a menor, u otros criterios como relevancia o alfabéticamente), para poder analizar primero las opciones que más se ajustan a mis prioridades (por ejemplo, buscar la opción más económica).

* **Criterios de Aceptación:**
* *Dado* un conjunto de resultados ya filtrados o sin filtrar, *cuando* el usuario selecciona un criterio de ordenamiento en la interfaz (por ejemplo, elige “Precio: menor a mayor” en un menú desplegable de “Ordenar por”), *entonces* la lista de resultados se reordena inmediatamente según ese criterio, reorganizando las tarjetas/filas de repuestos para reflejar el nuevo orden.
* Los criterios de orden disponibles deben incluir al menos: **Precio ascendente** (de barato a caro), **Precio descendente** (de caro a barato), y un orden por **Relevancia** (por defecto, cuando aplique una búsqueda textual). *Cuando* el usuario no ha seleccionado ninguno explícitamente, *entonces* se usa la relevancia o un orden predefinido por el sistema.
* *Dado* que el usuario cambia de criterio de orden múltiples veces, *cuando* aplica un nuevo criterio, *entonces* la lista se vuelve a ordenar instantáneamente sin necesidad de recargar completamente la página, idealmente con una pequeña animación o indicación visual para que note el cambio. El criterio actualmente activo queda resaltado o indicado (por ejemplo, aparece marcado en el menú de orden).
* *Dado* que la cantidad de resultados puede ser grande, *cuando* se aplica un ordenamiento, *entonces* el sistema debe seguir soportando la paginación o carga incremental correctamente en ese orden. Es decir, si son 100 resultados y se muestran 20 por página ordenados por precio, al ir a la página siguiente debe continuarse el orden global correctamente, solicitando al backend la siguiente página ya ordenada por ese criterio (el endpoint de búsqueda soporta un parámetro de sort).
* *Cuando* un criterio de orden no es aplicable o significativo para ciertos datos (por ejemplo, ordenar por relevancia en una vista que ya está filtrada por categoría podría equivaler al orden alfabético), *entonces* la interfaz puede ajustar o deshabilitar dicho criterio para evitar confusión. En todo momento, solo un criterio de orden puede estar activo simultáneamente.
* **Prioridad:** Could (Podría)

### HU-015: Como usuario, quiero que si **no se encuentran resultados** para mi búsqueda o si ocurre un error al buscar, la plataforma me muestre un mensaje claro y opciones para corregir o intentar de nuevo, para no quedarme sin orientación en esos casos.

* **Criterios de Aceptación:**
* *Dado* que el usuario realiza una búsqueda con términos muy específicos o inusuales, *cuando* ningún repuesto de la base coincide con la consulta, *entonces* la interfaz no mostrará una lista vacía abruptamente sino que presentará un **mensaje amigable** del tipo “No se han encontrado repuestos para ‘<término>’.”, posiblemente acompañado de sugerencias (ej.: “Prueba con otras palabras clave o verifica la ortografía”) o acciones alternativas (como un botón para volver a la búsqueda inicial).
* *Dado* que ocurre un problema de conexión o un error interno al realizar la búsqueda (por ejemplo, el API devuelve un error 500 o no responde), *cuando* el usuario espera los resultados, *entonces* la interfaz muestra una **notificación de error** indicando que hubo un problema técnico (ej.: “Error al recuperar los datos, por favor inténtalo nuevamente más tarde”) en un lenguaje sencillo. Además, podría ofrecer reintentar la búsqueda con un botón “Reintentar” una vez pasado un momento.
* *Cuando* la búsqueda no arroja resultados pero existen categorías relacionadas, *entonces* la interfaz podría sugerir navegar por categorías generales o populares como alternativa. Por ejemplo: “No encontramos resultados para ‘filtro XYZ’. Puedes explorar repuestos en la categoría *Filtros*.” Esta sugerencia se mostraría como enlaces a esas categorías, mejorando la experiencia ante resultados vacíos.
* *Dado* que el usuario ingresa accidentalmente caracteres inválidos o muy pocos caracteres (ej.: una sola letra) en la búsqueda, *cuando* intenta ejecutar la búsqueda, *entonces* la interfaz podría validar la entrada antes de enviarla y mostrar un mensaje corto como “Por favor ingresa al menos 2 caracteres para buscar” en vez de enviar la consulta que probablemente no arrojará nada. Esto previene también consultas inútiles al servidor.
* *Cuando* no hay resultados debido a filtros muy restrictivos (por ejemplo, el usuario puso un precio mínimo muy alto que eliminó todos), *entonces* en el mensaje de “sin resultados” también se indicará que podrían ajustarse o quitarse algunos filtros para obtener resultados (e.j., “Intenta ampliar el rango de precio o limpiar algunos filtros”). Esto guía al usuario para salir del estado vacío.
* **Prioridad:** Should (Debería)

## Épica 4: Funcionalidades de Comparación y Navegación (Experiencia de Usuario)

*Descripción:* Esta épica cubre las funcionalidades centrales de **comparación de productos** y la navegación hacia la información detallada de cada repuesto, incluyendo la salida del usuario hacia los sitios externos para concretar compras. Incluye la visualización de una página de **detalle** completo de un repuesto seleccionado, la capacidad de **comparar múltiples** repuestos lado a lado, la opción de **redirigir** al usuario al sitio original del producto para más información o compra, y características avanzadas como mostrar el **historial de precios** de un repuesto en su detalle. También contempla indicaciones de **disponibilidad** o estado de los repuestos (por ejemplo, si están agotados o si la información pudo quedar desactualizada), para mejorar la confianza del usuario en la información presentada. El objetivo es brindar al usuario herramientas para evaluar alternativas en profundidad y luego conectarlo con la fuente donde puede obtener el producto.

### HU-016: Como usuario, quiero ver el **detalle completo de un repuesto** cuando selecciono un resultado, incluyendo información ampliada (descripción, especificaciones, imágenes) y datos clave como precio actual, tienda origen y disponibilidad, para evaluar si cumple con mis necesidades antes de decidir una compra.

* **Criterios de Aceptación:**
* *Dado* un repuesto listado en los resultados de búsqueda, *cuando* el usuario hace clic en él (o toca, en dispositivo móvil), *entonces* la aplicación navega a una **página de detalle** dedicada para ese repuesto. En esta vista, se muestra el nombre completo del repuesto, una imagen ilustrativa de ser posible (por ejemplo, extraída o enlazada desde la fuente), la descripción detallada o ficha técnica (tamaño, modelo compatible, marca, etc. según el tipo de pieza), el precio actual y moneda, la tienda o portal de origen con su logo o nombre, la categoría a la que pertenece, y la última fecha de actualización de estos datos.
* *Cuando* se muestra la página de detalle, *entonces* también se incluyen elementos de interacción: por ejemplo, un botón destacado de **“Comprar en tienda”** o “Ver en sitio original” (ver HU-018), y si el usuario llegó aquí desde una comparación o búsqueda, un enlace para **volver** a la página anterior o resultados (mejorando la navegación).
* *Dado* que ciertos repuestos podrían no tener todos los campos disponibles (por ejemplo, quizás una fuente no provee descripción larga o imagen), *cuando* falte algún dato en el detalle, *entonces* la interfaz aún así mantiene un diseño limpio: omite ese campo o muestra un marcador “No disponible” discretamente, de manera que la ausencia de información no se confunda con un error. Por ejemplo, si no hay imagen, se muestra un placeholder ilustrativo; si no hay detalle técnico, se podría ocultar la sección de “Especificaciones”.
* *Cuando* la página de detalle carga información que está siendo obtenida del API (endpoints de HU-007 y HU-019), *entonces* se presenta un **indicador de carga** (loading spinner o similar) hasta que todos los datos estén listos, evitando que el usuario vea un espacio en blanco. Si la carga de datos falla (timeout o error del API), se muestra un mensaje de error en el cuerpo de la página de detalle indicándolo y se sugiere reintentar o volver atrás.
* **Prioridad:** Must (Obligatorio)

### HU-017: Como usuario, quiero **comparar varios repuestos** que he seleccionado, viéndolos lado a lado en la pantalla, para analizar sus diferencias en precio, características y otras especificaciones y así decidir cuál me conviene más.

* **Criterios de Aceptación:**
* *Dado* que en la lista de resultados cada repuesto tiene una opción para seleccionarlo (por ejemplo, una casilla “Comparar” o un ícono de checklist), *cuando* el usuario marca dos o más repuestos y luego activa la acción “Comparar” (por ejemplo, pulsando un botón fijo que dice “Comparar (3)” cuando ha seleccionado 3 items), *entonces* la aplicación navega a una vista de **comparación** donde se muestran las columnas con cada repuesto seleccionado.
* *Cuando* se muestra la tabla o cuadricula de comparación, *entonces* cada repuesto ocupa una columna y las filas representan atributos: por ejemplo, una fila para el nombre, otra para imagen, otra para precio, otra para tienda, otra para alguna especificación clave (dependiendo del tipo de producto). Los valores de cada repuesto están alineados para facilitar la comparación directa de una característica a través de todos los ítems.
* *Cuando* existan diferencias notables entre repuestos comparados (por ejemplo, uno es significativamente más barato, o uno es de una marca distinta), *entonces* la interfaz **destaca visualmente** esas diferencias: podría resaltar en verde el precio más bajo entre los comparados y en rojo el más alto, o poner en negrita las características únicas de un ítem. Esto ayuda al usuario a identificar de un vistazo las ventajas y desventajas de cada opción.
* *Dado* que un usuario podría intentar comparar pocos o demasiados items, *cuando* el usuario ha seleccionado menos de 2 repuestos, *entonces* el botón/acción de comparar permanecerá deshabilitado o al pulsarlo mostrará un mensaje “Seleccione al menos dos productos para comparar.”; y *cuando* ha seleccionado más del máximo soportado (ej.: más de 4), *entonces* el sistema podría limitar la comparación a 4 columnas y notificar “Has seleccionado demasiados productos, solo se compararán los primeros 4” o simplemente deshabilitar seleccionar más con una indicación de límite.
* *Cuando* el usuario está en la vista de comparación, *entonces* tiene la posibilidad de **remover** elementos de la comparación (por ejemplo, un botón “X” en la esquina de cada columna) o **agregar más** desde esa misma pantalla (quizás un campo de búsqueda adicional), aunque esto último podría redirigirlo de vuelta a la lista. Remover un ítem actualiza la vista inmediatamente reestructurando las columnas restantes.
* *Dado* que la vista de comparación debe ser usable en distintos tamaños de pantalla, *cuando* la comparación se visualiza en pantallas pequeñas (móvil), *entonces* en lugar de múltiples columnas estrechas, la interfaz puede permitir hacer *scroll* horizontal para ver todas las columnas o apilar parcialmente la información de cada repuesto, asegurando que la comparación siga siendo posible aunque el formato se adapte.
* **Prioridad:** Must (Obligatorio)

### HU-018: Como usuario, quiero **ser redirigido al sitio web original** donde se ofrece el repuesto que me interesa (por ejemplo, la tienda en línea o portal de clasificados) mediante un enlace claro, para poder realizar la compra o coordinar la adquisición directamente en la fuente original.

* **Criterios de Aceptación:**
* *Dado* que estoy viendo la información de un repuesto (en la vista de detalle HU-016 o incluso desde la lista de resultados si hay un acceso directo), *cuando* hago clic en el botón o enlace “Ver en la tienda” o “Comprar en sitio original”, *entonces* la aplicación abre la URL externa correspondiente al producto en cuestión. Esta apertura debe hacerse preferiblemente en una **nueva pestaña** o ventana del navegador, de modo que el usuario no pierda la página de CarMatch y pueda volver fácilmente después de revisar el sitio externo.
* *Cuando* se activa la redirección, *entonces* el sistema podría registrar ese evento (por ejemplo, para métricas internas de cuántos usuarios hacen clic hacia cada fuente, aunque esto sería invisible para el usuario) pero no bloquea ni interfiere en la navegación; la prioridad es llevar al usuario al destino lo más directo posible.
* *Dado* que el estado o disponibilidad del producto en la fuente original podría cambiar (por ejemplo, el enlace puede caducar si el producto fue retirado), *cuando* el usuario hace clic en “Ver en la tienda” y la página externa resulta en un **error 404 o producto no encontrado**, *entonces* desde CarMatch no se puede controlar la página externa, pero se podría presentar un mensaje previo o indicador si ya se sabía que ese ítem estaba marcado como agotado (ver HU-020). En caso contrario, el usuario verá el error en la página externa; CarMatch debería ofrecer un camino para reportar enlace caído (no obligatorio, pero podría ser un plus).
* *Cuando* por alguna razón la redirección no se pudiera realizar (ej.: bloqueo de pop-ups, o error en la URL formada), *entonces* la interfaz de CarMatch notifica al usuario con un mensaje del tipo “No fue posible abrir el sitio externo. Por favor habilita las ventanas emergentes o intenta desde este <a href='URL'>enlace directo</a>.”, para asistirlo a continuar.
* La acción de redirigir debe estar claramente identificada con texto e íconos (por ejemplo, un ícono de enlace externo/flecha) para que el usuario entienda que saldrá de CarMatch. Asimismo, *cuando* el cursor pasa sobre el botón, *entonces* se podría mostrar en tooltip la URL destino o el nombre de la tienda (ej.: “Ir a MercadoLibre.cl”) para mayor transparencia.
* **Prioridad:** Must (Obligatorio)

### HU-019: Como usuario, quiero visualizar el **historial de precios** de un repuesto en su página de detalle, para entender cómo ha variado su precio en el tiempo y decidir si me conviene comprarlo ahora o esperar una oferta.

* **Criterios de Aceptación:**
* *Dado* que el sistema ha ido almacenando los cambios de precio de un repuesto específico en cada actualización (scraping) exitosa, *cuando* el usuario abre la página de detalle de ese repuesto, *entonces* en alguna sección de la vista se muestra un **gráfico o listado** histórico de precios: por ejemplo, un gráfico de línea que eje X es la fecha y eje Y el precio, o bien una tabla cronológica de fechas y precios, indicando el precio inicial registrado y las variaciones hasta el actual.
* *Cuando* el historial de precios se muestra en formato gráfico, *entonces* el usuario puede visualizar claramente tendencias (subidas, bajadas, estabilidad). Además, podría interactuar con el gráfico pasando el mouse o tocando puntos para ver el valor exacto y fecha de ese punto específico (“10/09/2025 – $50.000”).
* *Dado* que un repuesto podría ser recién ingresado (poco tiempo de seguimiento) o que no se detectaron cambios de precio, *cuando* el historial no tenga suficientes datos, *entonces* la sección podría mostrar un mensaje como “No hay datos históricos suficientes” o simplemente mostrar un único punto indicando el precio actual con nota “Sin variaciones recientes”. Así el usuario entiende que no hay mucho histórico en lugar de pensar que no carga.
* *Cuando* el usuario observa el historial, *entonces* también se resalta el **precio actual** de alguna forma (por ejemplo, un punto destacado o anotación en el gráfico) para que pueda compararlo con precios pasados. Si el precio actual es el más bajo en cierto periodo, eso podría señalarse (ej.: texto “Precio mínimo de los últimos 3 meses”) para motivar la compra, o si ha subido recientemente, el usuario lo notará en la tendencia.
* *Dado* que los datos de historial vienen del backend (posiblemente del endpoint de HU-007 o uno específico), *cuando* haya cualquier problema en cargar el historial (p.ej. un fallo de red), *entonces* la sección debe manejarlo mostrando “Historial no disponible temporalmente” discretamente, sin impedir que el resto del detalle (que sí cargó) sea visible.
* **Prioridad:** Could (Podría)

### HU-020: Como usuario, quiero que la plataforma **indique si un repuesto ya no está disponible** o está desactualizado (por ejemplo, agotado en su tienda de origen), para evitar perder tiempo en compararlo o intentar comprar algo que no podré conseguir.

* **Criterios de Aceptación:**
* *Dado* que un repuesto estaba en la base de datos pero la última vez que se intentó actualizar no se encontró en la fuente (ej.: el scraper no lo halló porque fue eliminado o vendido), *cuando* el sistema identifica esa situación durante el scraping, *entonces* marca internamente ese item como **“inactivo”** o no disponible.
* *Cuando* un repuesto está marcado como inactivo/no disponible, *entonces* en la **lista de resultados** su presentación visual cambia: puede mostrarse atenuado (grayscale) o con una etiqueta “Agotado” sobre la imagen, y no se incluye su precio (o se indica “—” si no hay stock). Asimismo, si el usuario intenta seleccionarlo para comparar, se puede impedir la selección con un tooltip “Este producto ya no está disponible”.
* *Cuando* el usuario abre la **página de detalle** de un repuesto no disponible, *entonces* se muestra claramente un mensaje o distintivo (por ejemplo, un texto rojo “Este repuesto ya no está disponible en la tienda original”) y el botón de “Ver en la tienda” puede estar deshabilitado o oculto para no dirigir al usuario a un enlace muerto. Podría ofrecerse en su lugar un botón “Buscar similares” que lo lleve de vuelta a resultados de búsqueda con un filtro similar, como mejora opcional.
* *Dado* que la disponibilidad puede cambiar (por ejemplo, un producto agotado vuelve a estar en stock), *cuando* el sistema realiza nuevas pasadas de scraping, *entonces* si detecta que un repuesto marcado como inactivo reapareció o sigue ausente se actualiza su estado en consecuencia. Los usuarios siempre deben ver el estado más preciso posible basado en la última actualización.
* *Cuando* se genera la vista de comparación (HU-017) y uno de los productos comparados está no disponible, *entonces* igualmente aparece en la tabla pero resaltado como “No disponible” en su columna, para que el usuario sepa que esa opción quizás ya no puede adquirirse. Si todos los comparados menos uno están no disponibles, quizás se sugiere al usuario que solo el disponible es viable.
* **Prioridad:** Could (Podría)

## Épica 5: Panel de Administración y Métricas del Sistema

*Descripción:* Esta épica comprende las funcionalidades dirigidas al **administrador** de CarMatch, accesibles mediante un panel de control web seguro. Incluye la gestión de acceso (login/logout) de administradores, la visualización de **métricas** tanto de uso de la plataforma (búsquedas realizadas, comparaciones, clics de redirección) como del estado del sistema (cantidad de datos, actualizaciones, rendimiento), la consulta de **logs** de eventos importantes, y la configuración avanzada del sistema como la gestión de categorías unificadas. Además, considera la definición de mecanismos de **monitoreo y alertas** automáticas ante eventos críticos (como caídas o errores persistentes). Esta épica garantiza que el equipo administrador tenga las herramientas para mantener, supervisar y hacer crecer la plataforma de manera informada y segura.

### HU-021: Como administrador, quiero **autenticarme de forma segura** en un portal de administración mediante credenciales (usuario/contraseña) o single sign-on, para que solo personal autorizado pueda acceder a las configuraciones y datos sensibles del sistema.

* **Criterios de Aceptación:**
* *Dado* que el administrador navega a la URL del panel de administración (ej.: /admin), *cuando* ingresa un nombre de usuario y contraseña válidos y envía el formulario de login, *entonces* el sistema verifica las credenciales y, si son correctas, le permite el acceso mostrando la pantalla principal del panel admin (dashboard), estableciendo además una sesión segura o token para mantener al admin autenticado durante su interacción.
* *Dado* que el administrador ingresa credenciales incorrectas, *cuando* intenta autenticarse, *entonces* la interfaz muestra un mensaje de error claro (“Usuario o contraseña incorrectos”) sin especificar cuál de los dos está mal, por seguridad. Tras ciertos intentos fallidos (ej.: 5), podría aplicarse un breve bloqueo temporal de la cuenta o captcha para proteger contra fuerza bruta.
* *Cuando* el administrador está autenticado y ha permanecido inactivo por un tiempo prolongado (ej.: 30 minutos), *entonces* la sesión expira automáticamente y al intentar cualquier acción se lo redirige nuevamente a la pantalla de login, mostrando un mensaje “Sesión expirada, por favor ingrese de nuevo”, garantizando así que no quede la sesión abierta indebidamente.
* *Dado* que el panel muestra información sensible, *cuando* el admin inicia sesión correctamente, *entonces* la comunicación del login y del panel debe ocurrir sobre **conexión segura (HTTPS)**, y las contraseñas deben almacenarse de forma hasheada en la base de datos. Todo esto asegura que las credenciales no sean expuestas en texto plano y que el acceso al panel sea cifrado.
* *Cuando* un administrador desea salir, *entonces* tiene disponible un botón “Cerrar sesión” que al clic invalidará su token/sesión y lo llevará de vuelta a la pantalla de login, confirmando la acción. Tras logout, no puede volver atrás a páginas privadas sin re-login (se verifica esto).
* **Prioridad:** Must (Obligatorio)

### HU-022: Como administrador, quiero visualizar un **dashboard de métricas** con indicadores clave de la plataforma (por ejemplo, número de búsquedas realizadas en el día, repuestos más buscados, número de comparaciones hechas, clics de redirección, cantidad total de repuestos en base de datos, etc.), para poder **monitorizar el uso** y el rendimiento general del sistema de un vistazo.

* **Criterios de Aceptación:**
* *Cuando* el administrador accede al panel principal tras autenticarse, *entonces* se muestra un **dashboard** inicial con varios recuadros o gráficos que reflejan métricas en tiempo real o del último periodo relevante: por ejemplo, “Búsquedas realizadas hoy” (con un contador o gráfica de barras por hora), “Top 5 términos buscados esta semana”, “Comparaciones realizadas hoy”, “Clicks de ‘Ver en tienda’ hoy”, “Total de repuestos activos en la base de datos”, “Nuevos repuestos añadidos en la última actualización”, etc.
* *Dado* que muchas métricas son temporales, *cuando* el administrador lo requiera, *entonces* puede ajustar filtros de fecha en el dashboard (ej.: ver estadísticas de un rango específico, día anterior, último mes). Si selecciona un rango o fecha específica, los gráficos y contadores se actualizan para reflejar ese intervalo.
* *Cuando* una métrica específica tenga un valor que requiera atención (por ejemplo, 0 búsquedas en un día normal, lo que sería inusual, o un pico anormalmente alto de ciertos eventos), *entonces* ese recuadro podría resaltarse (en color diferente) o listar una pequeña alerta indicando que podría requerir revisión. Esto ayuda a detectar comportamientos anómalos.
* Las métricas deben presentarse de forma visualmente amigable: *cuando* hay datos comparativos (ej.: hoy vs ayer), *entonces* se pueden usar flechas o porcentajes mostrando la tendencia (ej.: “+20% vs día anterior” en verde si aumentó el uso). Gráficos simples (barras, líneas) acompañan a números clave para mejor interpretación.
* *Dado* que la consulta de métricas podría ser costosa si hay muchos datos históricos, *cuando* se cargue el dashboard, *entonces* las consultas deben estar optimizadas (agregaciones precomputadas, caching de métricas diarias) para que la página se muestre en pocos segundos. En caso de retraso en alguna métrica, se puede cargar asíncronamente mostrando un placeholder “Cargando…” en ese componente.
* **Prioridad:** Should (Debería)

### HU-023: Como administrador, quiero poder **consultar los logs** del sistema y el estado de los procesos clave (scraping, API, base de datos) desde el panel, para realizar tareas de **monitoreo manual** y diagnóstico cuando sea necesario.

* **Criterios de Aceptación:**
* *Cuando* el administrador navega a la sección “Logs y Estado del Sistema” en el panel, *entonces* se le presenta una interfaz con los registros recientes del sistema. Debe haber opción de elegir diferentes categorías de log o niveles (por ejemplo: *Eventos de scraping*, *Errores del servidor*, *Alertas de seguridad*) para facilitar la revisión de lo relevante en cada caso.
* *Dado* que los logs pueden ser muy extensos, *cuando* se muestran en el panel, *entonces* están paginados o con scroll interno, mostrando por ejemplo los últimos 100 eventos con posibilidad de cargar más. Cada entrada de log incluye al menos timestamp, nivel (info/warning/error), componente (ej.: “Scraper-FuenteX” o “API”), y mensaje. Los errores críticos podrían resaltarse en color.
* *Cuando* el administrador busca diagnosticar un problema específico, *entonces* tiene la posibilidad de **filtrar o buscar texto** dentro de los logs desde la interfaz (ej.: buscar por “FuenteX” para ver solo logs de esa fuente, o por “ERROR” para saltar a errores). Esto permite ubicar rápidamente eventos relevantes.
* Además de logs, la sección puede mostrar una síntesis del **estado actual**: *cuando* el admin abre esta vista, *entonces* se ven indicadores como “Última ejecución de scraping: OK, hace 2 horas”, “Estado de base de datos: Conectado (20k registros)”, “Uso de CPU del servidor: 75%” u otros parámetros de salud. Estos indicadores ofrecen un vistazo rápido para saber si todo funciona correctamente en ese momento.
* *Dado* que podría necesitarse exportar o ver en más detalle, *cuando* el admin lo requiera, *entonces* debería poder descargar un archivo de log completo (por ejemplo, último log diario) o acceder a logs históricos (seleccionar fecha), para análisis externos más profundos. Esto último es opcional pero útil para mantenimiento.
* **Prioridad:** Should (Debería)

### HU-024: Como administrador, quiero **gestionar un catálogo unificado de categorías** de repuestos y servicios en la plataforma, mapeando las distintas clasificaciones de las fuentes externas a categorías estándar internas, para asegurar que los filtros y la organización de la información sean consistentes para los usuarios.

* **Criterios de Aceptación:**
* *Cuando* el administrador accede a la sección de “Categorías” en el panel, *entonces* ve una lista de las categorías **estándar** que el sistema utiliza (por ejemplo: Motor, Transmisión, Eléctrico, Carrocería, Accesorios, Servicios, etc.), posiblemente organizadas jerárquicamente si aplica. Para cada categoría estándar, se puede ver cuántos repuestos están asignados a ella actualmente.
* *Dado* que las fuentes externas pueden llamar a las categorías de manera diferente (ej.: una fuente usa “Accesorios interiores” y otra “Interior” pero ambas caben en “Accesorios”), *cuando* el administrador selecciona una categoría estándar, *entonces* puede ver y editar el **mapeo** de categorías externas a esa categoría. Por ejemplo, asignar que “Accesorios interiores” (Fuente A) y “Interior” (Fuente B) correspondan a la categoría estándar “Accesorios”.
* *Cuando* se agrega una nueva fuente de datos (HU-002) que trae categorías no vistas antes, *entonces* el sistema debe permitir al administrador revisar estas categorías externas entrantes. Podría mostrar una lista de “Categorías sin mapear” detectadas en los datos y permitir asignarlas a una existente o crear una nueva categoría estándar si es necesario.
* *Dado* que la estructura de categorías influye en la experiencia de filtrado del usuario (HU-012), *cuando* el admin modifica el catálogo (por ejemplo, crea, fusiona o renombra una categoría estándar), *entonces* esos cambios se reflejan inmediatamente en el frontend: los filtros de categoría mostrarán los nuevos nombres o estructura. Asimismo, todos los repuestos asignados a la categoría antigua pasan a la nueva según la acción realizada (consistencia de datos).
* *Cuando* se realiza un cambio significativo (ej.: eliminar una categoría estándar), *entonces* el sistema solicita confirmación al administrador (por el impacto en datos asociados) y sólo la permite si no quedan repuestos mapeados, o alternativamente reasigna esos repuestos a una categoría “Por clasificar”. Así se evita dejar datos huérfanos sin categoría.
* **Prioridad:** Could (Podría)

### HU-025: Como administrador, quiero recibir **notificaciones de alerta** cuando ocurra algún evento crítico en el sistema (por ejemplo, que un scraping rutinario falle repetidamente, que el API esté caído, o que el uso de recursos supere cierto umbral), para poder tomar acciones proactivas sin tener que estar monitoreando constantemente.

* **Criterios de Aceptación:**
* *Dado* que un proceso crítico falla (por ejemplo, el scraping de *todas* las fuentes falló en una ejecución, o la API dejó de responder durante X minutos), *cuando* se detecta esta condición, *entonces* el sistema genera una **alerta** que es enviada al administrador a través de canales preconfigurados (por ejemplo, un correo electrónico automático detallando el evento, o una notificación push si existe una app/admin móvil).
* *Cuando* el administrador inicia sesión en el panel después de haberse generado una alerta, *entonces* en la interfaz verá un **banner o icono de notificación** resaltado (por ejemplo, un ícono de campana con contador) y al desplegarlo encontrará los avisos pendientes, como “Alerta: Falla en scraping de 5 fuentes – revisar configuración” o “Alerta: Uso de CPU > 90% durante 10 min”. Esto asegura que incluso si omitió el correo, lo vea en la plataforma.
* *Dado* que no todos los eventos requieren la misma urgencia, *cuando* el administrador configura las notificaciones (opción de configuración de alertas), *entonces* puede elegir qué tipos de eventos disparan alertas y a través de qué medio. Por ejemplo, marcar que sólo errores críticos envíen email inmediato, mientras que métricas de rendimiento envíen quizá un resumen diario.
* *Cuando* se resuelve un evento que causó la alerta (ej.: el sistema se recuperó, o el admin atendió el problema), *entonces* la alerta correspondiente en el panel cambia de estado (por ejemplo, de “activa” a “resuelta” o se archiva), manteniéndose para histórico pero sin figurar como pendiente. El admin también puede descartarla manualmente tras verla.
* *Dado* que puede haber falsos positivos o fluctuaciones breves que no ameritan alerta, *cuando* se diseñan las condiciones de disparo, *entonces* se incluyen criterios para evitar spam: por ejemplo, que el error persista N veces seguidas antes de alertar, o que el umbral de CPU alto se mantenga por Y minutos continuos. Esto no es visible para el usuario final pero importante en la implementación de la característica.
* **Prioridad:** Could (Podría)

## Matriz de Trazabilidad de Requisitos Funcionales ↔ Historias de Usuario

La siguiente tabla relaciona los **requisitos funcionales (RF)** identificados para el proyecto CarMatch con las historias de usuario que aseguran su implementación. De esta manera se garantiza la cobertura completa de los requerimientos en el backlog:

| Requisito Funcional (RF) | Historias de Usuario Asociadas |
| --- | --- |
| **RF-01:** El sistema debe recopilar información desde múltiples portales web de repuestos mediante procesos automatizados de **web scraping**. <br/>*(Incluye ejecución periódica sin intervención manual).* | HU-001, HU-005, HU-003 (log de fallos de scraping) |
| **RF-02:** El sistema debe permitir **gestionar las fuentes de datos** de scraping (agregar nuevas fuentes, editar configuración, desactivar/eliminar fuentes no deseadas). | HU-002, HU-005 |
| **RF-03:** El sistema debe **almacenar** los datos recolectados en una **base de datos** central en la nube (PostgreSQL en GCP), de forma estructurada, evitando duplicados y manteniendo consistencia. | HU-004, HU-010 |
| **RF-04:** El sistema debe exponer una **API REST** que provea acceso a los datos de repuestos para el frontend. Debe incluir endpoints para búsqueda con filtros, obtención de detalles y comparación de múltiples ítems. | HU-006, HU-007, HU-008 |
| **RF-05:** La plataforma debe ofrecer al **usuario final** una interfaz para **buscar repuestos** por palabra clave de manera rápida y sencilla. | HU-011, HU-015 |
| **RF-06:** La plataforma debe permitir **filtrar** los resultados de búsqueda por criterios relevantes (por ejemplo, categoría de producto, rango de precio, etc.) y **ordenarlos** según preferencia. | HU-012, HU-013, HU-014 |
| **RF-07:** La plataforma debe permitir al usuario **comparar múltiples repuestos** seleccionados simultáneamente, visualizando sus características lado a lado. | HU-017 |
| **RF-08:** La plataforma debe permitir al usuario acceder a la **información detallada** de un repuesto específico, incluyendo descripción completa, y desde allí **redirigir** al sitio externo original para su adquisición. | HU-016, HU-018 |
| **RF-09:** El sistema debe mantener **historial de cambios** (por ejemplo, historial de precios) de los repuestos y ofrecer esa información al usuario para apoyar la toma de decisiones. | HU-019 |
| **RF-10:** El sistema debe indicar la **disponibilidad** de los repuestos listados (por ejemplo, marcar los que estén agotados o no disponibles en origen). | HU-020 |
| **RF-11:** Debe haber un **panel de administración** restringido a usuarios autorizados, que permita configurar el sistema y visualizar su estado (incluyendo métricas de uso, registros de eventos y gestión de datos maestros como categorías). | HU-021, HU-022, HU-023, HU-024 |
| **RF-12:** El sistema debe incorporar mecanismos de **seguridad** y control: autenticación para el acceso de administradores, validación de entradas, protección contra accesos no autorizados o ataques comunes (OWASP), y políticas de uso seguro de la API. | HU-021, HU-009 |
| **RF-13:** El sistema debe contar con **monitoreo y alertas** que informen a los administradores sobre fallas o condiciones anómalas en la operación (fallos de scraping, caídas de servicios, altos usos de recursos), permitiendo acciones proactivas. | HU-003, HU-025 |
| **RF-14:** *(Extensión deseable)* El sistema debería unificar la taxonomía de **categorías** de productos de diferentes fuentes bajo un catálogo común, para mejorar la coherencia en la presentación y filtrado de la información. | HU-024 |

*(Nota: Los requisitos no funcionales —rendimiento, seguridad adicional, etc.— se listan en la sección de criterios RNF más abajo. RF-14 es considerado “podría tener”, incluido para robustez del diseño.)*

## Definition of Ready (DoR)

Antes de que una historia de usuario se considere **lista** para ser tomada por el equipo de desarrollo, debe cumplir con los siguientes criterios (Definition of Ready):

* La historia de usuario está **claramente redactada** siguiendo el formato establecido (rol, necesidad, beneficio) y es comprensible por todo el equipo; incluye todos los detalles necesarios para implementarla.
* **Criterios de aceptación** bien definidos y completos, redactados en formato BDD (Dado/Cuando/Entonces), que cubren casos positivos, negativos y alternativos según corresponda.
* Los **criterios de aceptación** cubren consideraciones de validación de datos, manejo de errores y estados especiales (vacíos, límites), de manera que la funcionalidad quede completamente especificada.
* La historia está **estimada** (p. ej. en story points) por el equipo, indicando que ha sido analizada y comprendida; su tamaño es adecuado (cumple INVEST: independiente, negociable, valiosa, estimable, pequeña, comprobable).
* Se han identificado y documentado cualquier **dependencia** externa o interna para implementar la historia (por ejemplo, necesidad de una API de terceros, definición previa de diseño UX, etc.) y estas dependencias están resueltas o planificadas.
* Cualquier duda o ambigüedad en la historia ha sido aclarada con el **Product Owner/Usuario** solicitante; hay consenso en el entendimiento de la funcionalidad.
* El **valor** aportado por la historia y su prioridad (MoSCoW) han sido validados por el Product Owner, asegurando que trabaja en las cosas correctas en el orden correcto.
* Los **criterios de aceptación** han sido revisados por el equipo de testing (QA) para asegurar que son comprobables y cubrirán los casos de prueba necesarios.
* Si la historia implica interfaces de usuario, se cuenta con un **diseño UI/UX** de referencia (wireframe o maqueta) aprobado, o al menos una descripción clara de cómo debe verse/interactuar, de modo que desarrolladores y testers tengan la misma visión.
* El equipo dispone de los **accesos o datos de entorno** necesarios (por ejemplo, credenciales de prueba para fuentes de datos externas, acceso a entornos de desarrollo configurados) para abordar la historia sin impedimentos.

En resumen, una historia de usuario **“Ready”** es aquella que está bien entendida, refinada y preparada para ser implementada sin información faltante que pueda trabar su desarrollo.

## Definition of Done (DoD)

Una historia de usuario se considera **“Done”** (terminada) cuando cumple con todos los criterios siguientes, asegurando la calidad y completitud de la entrega (Definition of Done):

* **Desarrollo completo**: El código implementa toda la funcionalidad descrita en la historia de usuario, cumpliendo todos sus criterios de aceptación. No quedaron tareas pendientes ni partes sin implementar.
* **Pruebas unitarias** desarrolladas y aprobadas: El módulo/código nuevo cuenta con pruebas unitarias (y/o de integración según corresponda) y todas pasan correctamente. El porcentaje de cobertura de pruebas se mantiene o mejora según el estándar del proyecto.
* **Pruebas de aceptación** superadas: Se ha verificado manualmente (y/o con pruebas automatizadas BDD) cada escenario de los criterios de aceptación en un entorno de prueba, confirmando que el comportamiento real coincide con lo especificado (incluyendo validaciones, mensajes de error, flujos alternos, etc.).
* **Sin defectos conocidos**: Cualquier bug encontrado durante el desarrollo o pruebas ha sido resuelto. No se arrastran defectos de severidad alta o media relacionados a esta funcionalidad. En caso de quedar alguna observación menor, el Product Owner ha sido informado y la ha aceptado para resolución posterior.
* **Código revisado**: Se realizó al menos una **code review** del código producido (por otro miembro del equipo), aplicando las guías de estilo y buenas prácticas. Los comentarios/recomendaciones de la revisión han sido atendidos.
* **Integración realizada**: La funcionalidad ha sido integrada al branch principal/release. Se ha verificado que no rompe la integración continua (build pasa exitosamente) ni introduce regresiones en otras partes del sistema (se corrieron tests de regresión si aplica).
* **Documentación actualizada**: Cualquier documentación relevante ha sido actualizada: por ejemplo, comentarios en el código, documentación de la API (p. ej., especificación Swagger/OpenAPI si el endpoint cambió o se agregó), manual de usuario o ayudas en la UI si aplica, README sobre pasos de despliegue si algo cambió, etc.
* **Deploy en ambiente de prueba**: La historia está desplegada en el ambiente de **staging/integración** para una validación final. El Product Owner u stakeholder ha podido revisar la funcionalidad allí y la ha aceptado conforme a los criterios.
* **Criterios no funcionales OK**: Se confirmó que la historia no violó ningún requisito no funcional importante: por ejemplo, se probó que el rendimiento es aceptable con la nueva funcionalidad, que la seguridad no se comprometió (p. ej., intentos de inyección en nuevos inputs), y que se mantiene la compatibilidad/responsividad de la UI.
* **Merge y cierre**: Tras aceptación, la historia (ticket) ha sido marcada como completada en el tablero, y el código correspondiente está preparado para ser incluido en la siguiente liberación/versión del producto.

Resumiendo, **“Done”** implica no solo codificación terminada, sino calidad verificada, impacto controlado en el sistema global, y valor entregado validado por el cliente/PO.

## Criterios de Requisitos No Funcionales (RNF)

Además de las historias funcionales, el sistema CarMatch debe cumplir con una serie de **requisitos no funcionales** que aseguran la calidad global de la solución en términos de rendimiento, seguridad, usabilidad, etc. A continuación, se listan los criterios RNF relevantes:

* **Rendimiento**: Las búsquedas y comparaciones deben responder rápidamente. El sistema debe manejar al menos *N* usuarios concurrentes realizando búsquedas sin degradación notable. *Criterio:* Las consultas típicas de búsqueda (HU-006) deberían completarse en <2 segundos en promedio con una base de datos de varios miles de registros. Se implementa *caché* en endpoints críticos (HU-008) y *paginación* para evitar transferir datos excesivos en una sola respuesta.
* **Escalabilidad**: La arquitectura debe soportar crecimiento en volumen de datos y tráfico. El uso de GCP (Google Cloud) para la base de datos y posibilidad de desplegar el backend en la nube permite escalar verticalmente u horizontalmente según demanda. *Criterio:* La base de datos está normalizada e indexada (HU-010) para crecer a decenas de miles de repuestos, y la aplicación puede escalar a múltiples nodos de scraping o instancias de API si la carga lo requiere.
* **Seguridad**: Protección de datos y control de acceso son primordiales. *Criterios:* Toda comunicación se realiza bajo HTTPS para proteger datos en tránsito. La API implementa autenticación JWT o sesión segura para administradores (HU-021, HU-009) y valida exhaustivamente inputs para prevenir inyecciones SQL, XSS y CSRF. Se establece limitación de tasa (rate limiting) en la API pública (HU-009) para mitigar abusos. Las credenciales y datos sensibles (por ejemplo, keys de APIs de terceros o contraseñas) nunca se almacenan en texto plano.
* **Disponibilidad & Confiabilidad**: El sistema debe ser robusto ante fallos. *Criterios:* Tareas críticas de scraping están monitorizadas y en caso de error no detienen toda la aplicación (HU-001, HU-003). Se configuran **backups automáticos** periódicos de la base de datos (por ejemplo, diarios) y se cuenta con un plan de recuperación de desastres (DRP) para restaurar el servicio rápidamente ante una caída grave. Un monitoreo externo de uptime podría alertar si la página o API caen.
* **Mantenibilidad**: El código y la configuración deben ser claros y manejables. *Criterios:* Se sigue un estilo de codificación consistente; existen módulos separados para scraping, backend, frontend respetando responsabilidades (se favorece la arquitectura modular). Hay documentación técnica (diagrama de arquitectura, diccionario de datos, comentarios en puntos complejos) que facilita a nuevos desarrolladores entender el sistema. La infraestructura en GCP y N8N está configurada vía scripts o documentación reproducible.
* **Usabilidad & UX**: La interfaz debe ser intuitiva y agradable para el usuario final. *Criterios:* El frontend Next.js implementa diseños responsive y accesibles (HU-011, HU-017) cumpliendo con las pautas **WCAG 2.1 AA** en lo posible (contrastes adecuados, navegación con teclado, textos alternativos en imágenes). La experiencia es consistente: se utilizan mensajes claros para errores (HU-015) y estados vacíos, y se proporcionan ayudas visuales (por ejemplo, placeholders, tooltips, indicadores de carga) para guiar al usuario. Pruebas de usabilidad básicas se realizan internamente para refinar la interacción.
* **Compatibilidad**: La aplicación web debe funcionar correctamente en los principales navegadores modernos (Chrome, Firefox, Safari, Edge) y en distintas resoluciones (desktop, tablets, móviles). *Criterio:* Se hacen pruebas en al menos los navegadores con mayor cuota en Chile; el diseño responsive garantiza uso desde smartphones sin pérdida de funcionalidad (HU-011, HU-017 consideraron eso).
* **Integridad de datos**: Los datos mostrados al usuario deben ser confiables y consistentes. *Criterios:* Se implementan verificaciones para evitar registros duplicados (HU-004) y mantener datos normalizados. Si ocurren errores en scraping, no se introducen datos corruptos; más bien se notifica al admin (HU-003) y los datos anteriores se conservan hasta nueva actualización correcta. Cualquier cambio importante en la estructura de datos requiere migraciones planificadas para no romper la aplicación.
* **Auditoría y Trazabilidad**: Debe haber registro de acciones importantes para seguimiento. *Criterio:* El sistema de logging (HU-003, HU-023) almacena eventos significativos (login de admin, cambios en fuentes, errores, etc.), con timestamps y detalles, permitiendo auditorías de qué ocurrió en el sistema y facilitando debugging en caso de incidentes.

Estos requisitos no funcionales complementan las historias de usuario, asegurando que no solo **qué** hace el sistema es correcto, sino también **cómo** lo hace en términos de calidad, seguridad y performance, alineándose con las expectativas de un producto robusto y profesional.

## Priorización MoSCoW (Actualizada)

A continuación se resume la prioridad MoSCoW de todas las historias de usuario en el backlog, incorporando las nuevas historias añadidas. Esta priorización orienta la orden de implementación, distinguiendo elementos imprescindibles de los opcionales:

* **Must Have (Imprescindibles):** Historias de valor crítico que constituyen el núcleo funcional mínimo de CarMatch y deben implementarse sí o sí:
* HU-001 (Scraping automático programado de fuentes de datos principales).
* HU-003 (Registro de errores críticos en scraping, asegurando continuidad del proceso).
* HU-006 (Endpoint API de búsqueda de repuestos con filtros básicos).
* HU-009 (Seguridad básica de la API y autenticación/admin, protege acceso).
* HU-010 (Persistencia de datos en PostgreSQL, modelo base estable).
* HU-011 (Función de búsqueda en frontend y listado de resultados).
* HU-016 (Vista de detalle de repuesto, información completa disponible al usuario).
* HU-017 (Función de comparación de múltiples repuestos, esencia comparativa de la plataforma).
* HU-018 (Redirección al sitio externo para concretar la compra, cierre del ciclo de uso).
* HU-021 (Login/Autenticación de administrador para el panel, seguridad de backend).
* **Should Have (Debería tener):** Historias importantes que aportan gran valor o mejoran la experiencia, aunque no absolutamente vitales para un MVP básico. Deberían incluirse si es posible en la primera versión funcional:
* HU-002 (Gestión de fuentes de datos en interfaz – facilita ampliar/ajustar cobertura sin tocar código).
* HU-004 (Eliminación de duplicados y normalización de datos – mejora calidad de información presentada).
* HU-012 (Filtro por categoría en resultados – ayuda a refinar búsquedas).
* HU-013 (Filtro por rango de precio – muy útil para usuarios con presupuesto).
* HU-015 (Manejo de casos sin resultados/errores en búsqueda – mejora usabilidad y percepción de calidad).
* HU-022 (Dashboard de métricas de uso – proporciona visibilidad del funcionamiento y valor generado).
* HU-023 (Visualización de logs/estado – para monitoreo manual y soporte).
* HU-007 (Endpoint de detalle – útil para modularizar datos enviados al frontend, aunque en MVP podría cargarse todo en búsqueda).
* **Could Have (Podría tener):** Historias deseables que aportarían valor añadido o sofisticación, pero cuyo no cumplimiento no invalida el funcionamiento básico. Se implementarán si el tiempo y recursos lo permiten, posiblemente en iteraciones posteriores a la versión inicial:
* HU-005 (Configuración avanzada de frecuencia de scraping y ejecuciones manuales – optimiza recursos y control fino).
* HU-008 (Endpoint de comparación con caché – mejora rendimiento en comparativas, pero la lógica podría resolverse en frontend de forma temporal).
* HU-014 (Ordenamiento de resultados – conveniencia de UX, la plataforma podría operar inicialmente sin ordenamientos complejos).
* HU-019 (Historial de precios en detalle – información extra valiosa para usuarios avanzados, pero no esencial para encontrar y comparar productos estáticamente).
* HU-020 (Indicador de disponibilidad/agotado – mejora confianza del usuario en datos, pero MVP podría no detectarlo en tiempo real).
* HU-024 (Gestión de categorías unificadas – importante para escalabilidad con muchas fuentes, pero si inicialmente son pocas fuentes, se puede manejar manualmente la consistencia).
* HU-025 (Notificaciones de alerta proactivas – útil para operación 24/7, pero al inicio el monitoreo podría ser manual via revisión de logs hasta consolidar el sistema).
* **Won’t Have (No se hará por ahora):** Aspectos fuera del alcance de esta fase del proyecto. Pueden reevaluarse en futuras iteraciones, pero no serán implementados en la presente:
* **Registro/login de usuarios finales** ni funcionalidades de perfil (favoritos, alertas personales): La plataforma en esta etapa será de uso abierto sin cuentas de cliente; el foco está en la comparación anónima de repuestos.
* **Transacciones de compra dentro de CarMatch**: No se implementará carrito de compra propio ni pasarelas de pago; la compra se realiza en los sitios externos originales (CarMatch actúa solo como comparador y redirigidor).
* **Aplicación móvil nativa**: El alcance se limita a la aplicación web responsive. Una app móvil dedicada se considera fuera de esta entrega.
* **Integraciones con redes sociales o funcionalidades sociales** (compartir listas, reseñas de usuarios, etc.): No contemplado en el MVP, enfocado puramente en la búsqueda y comparación objetiva de productos.
* **Soporte multi-idioma**: La plataforma se centra en el mercado chileno en español por ahora; no se incorporará internacionalización en esta fase.

Con esta priorización MoSCoW, el equipo puede enfocar sus esfuerzos en entregar primero lo **Must** y **Should**, asegurando un producto funcional y de valor, mientras que los **Could** se planifican condicionalmente y los **Won’t** quedan explícitamente fuera del compromiso actual. Esto permite manejar expectativas y realizar cortes verticales del producto de manera iterativa e incremental, garantizando que las funcionalidades críticas estén presentes a tiempo y con la calidad esperada.