Regression Test Documentation eRoom-Tv

Regression Test

Informe final: **Modelo de Informe de Regression Test.odt**

El Regression Test tiene como objetivo:

Verificar que las funcionalidades existentes siguen comportándose igual que antes de los cambios.

Detectar errores introducidos por optimizaciones, refactorizaciones o correcciones de bugs.

Asegurar que las modificaciones no reintroduzcan fallos antiguos (reversiones de fallos).

Confirmar que las correcciones de defectos recientes no afecten negativamente a otras partes del sistema.

Mantener la estabilidad general del software a lo largo del tiempo.

Pasos para realizar un test de regresión en QA.

1. **Definir alcance:**

Módulos a trabajar en el test de regresión: Cast list Pairs list, Rooms list, Ads list, Look and feel, Channels list, Booking list.

2. funcionalidades y escenarios cubrir:

- ☐ filtros de búsqueda y los botones de descarga CSV
- ☑ paginado en el modulo Channels list
- ☐ infomacion del Archivo CSV módulo dispositivos
- ☐ Mejora en el comportamiento cuando cambio de idioma en el tv android

☐ Corregir error de zona horaria en la programación de publicidad para TV Android

☐ Implementar filtro por estado en el módulo de Reservas (Booking)

☐ Implementar en el menor tiempo los cambios realizados desde la consola de administración.

☐

Remover imágenes por defecto en app TV

☐ Foto de backgrounds rota.

☑ El widget de información del hotel se muestra en blanco.

☑ El widget Live TV redirecciona a pluto tv

☐ Revisar configuracion de volumen de tv e-room

☐ Fallo de transmisión del widget de cast desde el TV

☐ Fallo de transmisión del widget de cast desde el TV

☐ Corregir imágenes de publicidad que se salen de los márgenes en TV Android

3. Identificar riesgos y áreas críticas.

☐ Live tv.

☐ Paginado en modulos requeridos.

☐ look and feel

☐ Correcciones de Widget.

☐ Tiempos de actualizaciones.

☐ Configuraciones del Tv.

4. Priorizar casos de prueba basados en impacto y probabilidad de fallo.

ID	Resumen	Prioridad	Tipo	Estado
ERTV-187	Dashboard-Realtime lógica de la tarjeta "Emparejar Dispositivos"	Normal	Error	Backlog
ERTV-221	Implementar en el menor tiempo los cambios realizados desde la consola de administración.	Mayor	Error	Backlog
~~ERTV-219~~	Foto de backgrounds rota.	Mayor	Error	Won't fix
~~ERTV-217~~	El widget de información del hotel se muestra en blanco.	Mayor	Error	Won't fix
~~ERTV-216~~	El widget Live TV redirecciona a pluto tv	Mayor	Error	Won't fix
ERTV-223	La app Rakuten Tv genera error	Normal	Error	Backlog
ERTV-224	La app Disney requiere actualización	Normal	Error	Backlog
ERTV-226	La app TDT channels requiere actualización	Normal	Error	Backlog
ERTV-229	La app movistar tv genera un error	Normal	Error	Backlog
~~ERTV-234~~	Videos de Publicidad tv no se visualizan correctamente	Mayor	Error	Won't fix
ERTV-246	Revisar configuracion de volumen de tv e-room	Mayor	Error	Backlog
ERTV-248	Fallo de transmisión del widget de cast desde el TV	Crítico	Error	Backlog
~~ERTV-249~~	Arreglar widget de clima que no muestra datos reales	Mayor	Error	Won't fix
ERTV-264	Revisar la infomacion del Archivo CSV módulo dispositivos	Mayor	Error	Backlog
ERTV-184	Dashboard-Realtime	Baja	Tarea	Backlog
ERTV-185	Alinear correctamente los filtros de búsqueda y los botones de descarga CSV y reiniciar Cast	Normal	Tarea	Backlog
ERTV-220	Remover imágenes por defecto en app TV	Baja	Tarea	Backlog
ERTV-232	Se requiere paginado en el modulo Channels list	Mayor	Tarea	Backlog

ERTV-245	Botón avanzado fuera del espacio de menú	Normal	Tarea	Backlog
ERTV-265	Revisar el funcionamiento del Filtro del módulos: Dispositivos/Pair/Habitaciones	Mayor	Tarea	Backlog
ERTV-268	Mejora en el comportamiento cuando cambio de idioma en el tv android	Normal	Tarea	Backlog
ERTV-269	Corregir error de zona horaria en la programación de publicidad para TV Android	Crítico	Tarea	Backlog
ERTV-270	Corregir imágenes de publicidad que se salen de los márgenes en TV Android	Mayor	Tarea	Backlog
~~ERTV-275~~	Corregir superposición de código QR y la hora en la publicidad de TV Android	Normal	Tarea	Won't fix
ERTV-276	Implementar filtro por estado en el módulo de Reservas (Booking)	Mayor	Tarea	Backlog
~~ERTV-277~~	Revisar visualización de publicidad en el Cast	Normal	Tarea	Ready
ERTV-37	eTV- Avance General - Desarrollo Continuo TV Admin - Core	Normal	Épica	In Process

5. Establecer criterios de aceptación y umbrales de calidad. (SE REQUIERE VERIFICAR CON EL EQUIPO DE DESARROLLO).

1. Establecer criterios de aceptación y umbrales de calidad.

1) Criterios de aceptación funcional (nivel de sistema)

- **Acceso y disponibilidad**
  - El sistema web debe estar accesible 99.9% del tiempo.
  - Pautas de mantenimiento: ventanas planificadas fuera de horario comercial reducidas y notificadas con anticipación.
- **Rendimiento end-to-end**
  - Carga de la página de inicio en ≤ 3 segundos en conexión promedio (p 95 en producción).

    - Tiempos de respuesta de API (back-end) dentro de los límites establecidos (ver umbrales de calidad). **(SE REQUIERE VERIFICAR CON EL EQUIPO DE DESARROLLO).**
    - El rendering de menús en la página clave debe completar en ≤ 2 segundos en dispositivos de gama media. **(SE REQUIERE VERIFICAR CON EL EQUIPO DE DESARROLLO).**
- **Funcionalidad principal**
    - Visualización de menús: listado, filtros por categoría/acomodaciones, búsqueda, y ordenamiento deben funcionar sin errores.
    - Detalle de menú: página de detalle con todos los campos obligatorios visibles y con enlaces a operaciones relacionadas.
- **Gestión de contenido**
    - Crear/editar/eliminar menús y categorías desde la interfaz administrativa con validación de campos y permisos correspondientes.
    - Actualización en tiempo real de disponibilidad y precios cuando corresponda.
    - Importación/exportación de datos de menús en formatos estándar QR.
- **Autenticación y autorización**
    - Inicio de sesión funcional para usuarios y roles (admin, staff, usuario).
    - Roles correctamente aplicados: acciones sensibles restringidas según permisos.
    - Revocación de sesiones y manejo de tokens (renovación/expiración).
- **Accesibilidad y usabilidad**
    - Soporte básico de accesibilidad (contraste, navegación por teclado, textos alternativos de imágenes).
    - Rendimiento razonable en dispositivos móviles (responsivo) y navegadores modernos.
- **Seguridad**
    - Protección contra inyección básica, validación de entradas, y manejo seguro de tokens.
    - Cifrado de datos sensibles en tránsito (TLS) y buenas prácticas de seguridad.
- **Observabilidad y mantenimiento**
    - Logs estructurados y trazabilidad por operación (ID de pedido/consulta).
    - Monitoreo de uptime, latencias y errores con alertas básicas (Slack/Email).
    - Pruebas de regresión automatizadas ejecutadas en cada despliegue.

2) Umbrales de calidad (target) REVISAR ESTE PUNTO CON EL EQUIPO DE DESARROLLO.

- **Latencia y rendimiento**
    - UI inicial: p95 ≤ 2.5 segundos; p99 ≤ 4 segundos en staging; en producción, p95 ≤ 3 segundos.
    - API crítica (GET /menus, GET /menus/{id}): p95 ≤ 500 ms en staging; ≤ 350 ms en producción.
    - Operaciones mutation (POST/PUT/DELETE): p95 ≤ 800 ms.
    - Tiempos de renderizado en movil: tiempo total de interacción inicial ≤ 4 segundos.
- **Errores y confiabilidad**
    - Tasa de error global (5xx) < 0.5% en producción; 4xx < 1% para usos válidos.
    - Latencia de p99 durante picos no exceder 2x de p95.

- **Integridad de datos**
  - 100% de operaciones de creación/actualización con validación de esquema y reglas de negocio.
  - Consistencia entre UI y back-end: datos mostrados reflejan estado real (disponibilidad, precios, etc.).
  - Sin datos sensibles expuestos en respuestas.
- **Seguridad y cumplimiento**
  - Pruebas de autenticación/ autorización cubiertas; fallos de autorización devuelven 401/403 correctamente.
  - Escaneo de vulnerabilidades sin fallos críticos en pipelines de CI/CD.
  - Cumplimiento básico de privacidad y retención de datos según políticas.
- **Calidad de contrato y pruebas**
  - Contratos API (OpenAPI/GraphQL) cubiertos al 100% con pruebas automatizadas.
  - Pruebas de regresión automatizadas ejecutadas en cada despliegue; tasa de fallo inferior al 1%.
- Experiencia de desarrollador/operaciones

Preparación del entorno y datos.

1. Asegurar un entorno aislado para pruebas (sandbox/staging) con datos consistentes.

   https://tv.uat.eroomsuite.com
2. Crear conjuntos de datos representativos: positivos, negativos, límites y casos de error (403, 404, 429, 500).

Selección y mantenimiento de casos de prueba REVISAR ESTE PUNTO CON EL EQUIPO DE DESARROLLO.

1. Casos de regresión para operaciones CRUD básicas, autenticación, autorización, validación de entradas y manejo de errores.
2. Pruebas de integración entre servicios (por ejemplo, API A llama a API B).
3. Pruebas de rendimiento y límites (p. ej., picos de tráfico simulados).

Ejecución de pruebas REVISAR ESTE PUNTO CON EL EQUIPO DE DESARROLLO.

1. Ejecutar en el orden definido, priorizando endpoints críticos.
2. Verificar código de estado correcto, estructura del cuerpo, cabeceras y tiempos de respuesta.
3. Registrar resultados y evidencia (logs, respuestas completas, trazas).
4. Documentar incidencias con pasos reproducibles y configuraciones de entorno.

Gestión de incidencias y seguimiento

1. Crear tickets de defecto con severidad, entorno, reproducibilidad y impacto.
2. Asignar responsables y plazos; agrupar fallos por componente.
3. Re-ejecutar pruebas críticas después de correcciones y cambios de API.

4. Mantener trazabilidad entre requerimientos, casos de prueba y fallos.

Validación de cambios y cierre

1. Verificar que las correcciones no impacten otras rutas (regresión cruzada).
2. Confirmar que se cumplen criterios de aceptación (latencia, código de respuesta, formato).
3. Generar reporte de regresión específico de API (resumen, cobertura por endpoint, métricas).
4. Aprobar cierre del ciclo de regresión.

Métricas y mejora continua

1. Cobertura de regresión de API (endpoint críticos y contratos).
2. Tasa de fallo por servicio, tiempo de ciclo de regresión, tasa de re-trabajo.
3. Lecciones aprendidas y mejoras en contratos, validación de esquemas y pruebas de rendimiento.

El Regression Test tiene como objetivo:

Verificar que las funcionalidades existentes siguen comportándose igual que antes de los cambios.

Detectar errores introducidos por optimizaciones, refactorizaciones o correcciones de bugs.

Asegurar que las modificaciones no reintroduzcan fallos antiguos (reversiones de fallos).

Confirmar que las correcciones de defectos recientes no afecten negativamente a otras partes del sistema.

Mantener la estabilidad general del software a lo largo del tiempo.

1. **Definir alcance:**

Módulos a trabajar en el test de regresión: **Pair, habitaciones, Ads, look and feel, channels, reservas.**

2. funcionalidades y escenarios cubrir:

3. Identificar riesgos y áreas críticas.

- [ ] Sección de crear carta/Platos.
- [ ] Validación en los tipos de idioma.
- [ ] En el modal de detalle del plato

- [ ] Carpetas DAM.
- [ ] Validación de la programación de cartas.
- [ ] Carga de imagenes en los platos.

1. **Priorizar casos de prueba basados en impacto y probabilidad de fallo.**

   Alinear correctamente los filtros de búsqueda y los botones de descarga CSV y reiniciar Cast.

   Dashboard-Realtime lógica de la tarjeta "Emparejar Dispositivos"

   Actualización de las app en el tv android

   paginado en el módulo Channels list

   Archivo CSV módulo dispositivos

   funcionamiento del Filtro del módulos: Dispositivos/Pair/Habitaciones

   Error de zona horaria en la programación de publicidad para TV Android

   filtro por estado en el módulo de Reservas (Booking)

   Remover imágenes por defecto en app TV

   Videos de Publicidad tv no se visualizan correctamente

   Revisar configuración de volumen de tv e-room

   Fallo de transmisión del widget de cast desde el TV

   Arreglar widget de clima que no muestra datos reales

   Corregir imágenes de publicidad que se salen de los márgenes en TV Android

2. Establecer criterios de aceptación y umbrales de calidad.

1) Criterios de aceptación funcional (nivel de sistema)

- **Acceso y disponibilidad**
  - El sistema web debe estar accesible 99.9% del tiempo.
  - Pautas de mantenimiento: ventanas planificadas fuera de horario comercial reducidas y notificadas con anticipación.
- **Rendimiento end-to-end**
  - Carga de la página de inicio en ≤ 3 segundos en conexión promedio (p 95 en producción).
  - Tiempos de respuesta de API (back-end) dentro de los límites establecidos (ver umbrales de calidad). **(SE REQUIERE VERIFICAR CON EL EQUIPO DE DESARROLLO).**
  - El rendering de menús en la página clave debe completar en ≤ 2 segundos en dispositivos de gama media. **(SE REQUIERE VERIFICAR CON EL EQUIPO DE DESARROLLO).**
- **Funcionalidad principal**
  - Visualización de menús: listado, filtros por categoría/acomodaciones, búsqueda, y ordenamiento deben funcionar sin errores.

    - Detalle de menú: página de detalle con todos los campos obligatorios visibles y con enlaces a operaciones relacionadas.
- **Gestión de contenido**
    - Crear/editar/eliminar menús y categorías desde la interfaz administrativa con validación de campos y permisos correspondientes.
    - Actualización en tiempo real de disponibilidad y precios cuando corresponda.
    - Importación/exportación de datos de menús en formatos estándar QR.
- **Autenticación y autorización**
    - Inicio de sesión funcional para usuarios y roles (admin, staff, usuario).
    - Roles correctamente aplicados: acciones sensibles restringidas según permisos.
    - Revocación de sesiones y manejo de tokens (renovación/expiración).
- **Accesibilidad y usabilidad**
    - Soporte básico de accesibilidad (contraste, navegación por teclado, textos alternativos de imágenes).
    - Rendimiento razonable en dispositivos móviles (responsivo) y navegadores modernos.
- **Seguridad**
    - Protección contra inyección básica, validación de entradas, y manejo seguro de tokens.
    - Cifrado de datos sensibles en tránsito (TLS) y buenas prácticas de seguridad.
- **Observabilidad y mantenimiento**
    - Logs estructurados y trazabilidad por operación (ID de pedido/consulta).
    - Monitoreo de uptime, latencias y errores con alertas básicas (Slack/Email).
    - Pruebas de regresión automatizadas ejecutadas en cada despliegue.

2) Umbrales de calidad (target) REVISAR ESTE PUNTO CON EL EQUIPO DE DESARROLLO

- **Latencia y rendimiento**
    - UI inicial: p95 ≤ 2.5 segundos; p99 ≤ 4 segundos en staging; en producción, p95 ≤ 3 segundos.
    - API crítica (GET /menus, GET /menus/{id}): p95 ≤ 500 ms en staging; ≤ 350 ms en producción.
    - Operaciones mutation (POST/PUT/DELETE): p95 ≤ 800 ms.
    - Tiempos de renderizado en movil: tiempo total de interacción inicial ≤ 4 segundos.
- **Errores y confiabilidad**
    - Tasa de error global (5xx) < 0.5% en producción; 4xx < 1% para usos válidos.
    - Latencia de p99 durante picos no exceder 2x de p95.
- **Integridad de datos**
    - 100% de operaciones de creación/actualización con validación de esquema y reglas de negocio.
    - Consistencia entre UI y back-end: datos mostrados reflejan estado real (disponibilidad, precios, etc.).
    - Sin datos sensibles expuestos en respuestas.
- **Seguridad y cumplimiento**
    - Pruebas de autenticación/ autorización cubiertas; fallos de autorización devuelven 401/403 correctamente.

    - Escaneo de vulnerabilidades sin fallos críticos en pipelines de CI/CD.
    - Cumplimiento básico de privacidad y retención de datos según políticas.
- **Calidad de contrato y pruebas**
    - Contratos API (OpenAPI/GraphQL) cubiertos al 100% con pruebas automatizadas.
    - Pruebas de regresión automatizadas ejecutadas en cada despliegue; tasa de fallo inferior al 1%.
- Experiencia de desarrollador/operaciones

Preparación del entorno y datos.

1. Asegurar un entorno aislado para pruebas (sandbox/staging) con datos consistentes.
2. Crear conjuntos de datos representativos: positivos, negativos, límites y casos de error (403, 404, 429, 500).
3. Configurar variables de entorno (endpoints, tokens, claves de API) para reproducibilidad.
4. Registrar versiones/buildes de la API y dependencias (microservicios relacionados).

3) Selección y mantenimiento de casos de prueba

1. Incluir pruebas de contrato/validación de esquema (ej. OpenAPI/ GraphQL schema).
2. Casos de regresión para operaciones CRUD básicas, autenticación, autorización, validación de entradas y manejo de errores.
3. Pruebas de integración entre servicios (por ejemplo, API A llama a API B).
4. Pruebas de rendimiento y límites (p. ej., picos de tráfico simulados).

4) Automatización (si aplica)

1. Diseñar tests automatizados que sean idempotentes y aislados (uso de datos independientes).
2. Utilizar herramientas adecuadas (Postman/Newman, REST-assured, JUnit/TestNG, PyTest, k6 para rendimiento).
3. Validar respuestas con esquemas JSON/YAML y aserciones de código de estado y payload.
4. Integrar ejecuciones en CI/CD (por ejemplo, cada push/PR dispara la suite).

5) Ejecución de pruebas

1. Ejecutar en el orden definido, priorizando endpoints críticos.
2. Verificar código de estado correcto, estructura del cuerpo, cabeceras y tiempos de respuesta.
3. Registrar resultados y evidencia (logs, respuestas completas, trazas).
4. Documentar incidencias con pasos reproducibles y configuraciones de entorno.

6) Gestión de incidencias y seguimiento

1. Crear tickets de defecto con severidad, entorno, reproducibilidad y impacto.
2. Asignar responsables y plazos; agrupar fallos por componente.
3. Re-ejecutar pruebas críticas después de correcciones y cambios de API.
4. Mantener trazabilidad entre requerimientos, casos de prueba y fallos.

7) Validación de cambios y cierre

1. Verificar que las correcciones no impacten otras rutas (regresión cruzada).
2. Confirmar que se cumplen criterios de aceptación (latencia, código de respuesta, formato).
3. Generar reporte de regresión específico de API (resumen, cobertura por endpoint, métricas).
4. Aprobar cierre del ciclo de regresión.

8) Métricas y mejora continua

1. Cobertura de regresión de API (endpoint críticos y contratos).
2. Tasa de fallo por servicio, tiempo de ciclo de regresión, tasa de re-trabajo.
3. Lecciones aprendidas y mejoras en contratos, validación de esquemas y pruebas de rendimiento.

Consejos prácticos para API

- Usa validación de esquemas estricta (JSON Schema, OpenAPI) para validar respuestas.
- Implementa pruebas de contrato entre microservicios para detectar rupturas de API.
- Prueba autenticación/autorización exhaustivamente (token expiration, scopes, RBAC).
- Incluye pruebas de manejo de errores y mensajes consistentes.
- Emplea entornos de prueba con datos aislados para no afectar producción.

Revision #12
Created 2025-08-26 16:38:23 UTC by Victoria Georgieff
Updated 2025-10-31 14:44:55 UTC by Victoria Georgieff