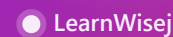


Preparación para producción, resolución de problemas y proyecto final

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Prepara las integraciones para aplicaciones reales: rendimiento, diagnóstico, actualizaciones en segundo plano, empaquetado, versionado y mantenibilidad. Qué significa esto para un desarrollador de integraciones con Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Prepara las integraciones para aplicaciones reales: rendimiento, diagnóstico, actualizaciones en segundo plano, empaquetado, versionado y mantenibilidad.

1. El problema de integración

Un widget puede funcionar en una demostración y aun así fallar en producción por condiciones de carrera, rutas de recursos rotas, un filtrado de eventos deficiente, falta de limpieza, bloqueos en tiempo de diseño o volúmenes de datos no probados.

2. Conceptos clave

Concepto	Cómo usarlo
Diagnóstico	Haz visibles los fallos mediante registros, herramientas y comprobaciones repetibles.
Eliminación	Limpia los recursos del cliente y los manejadores de eventos cuando se destruyen los widgets.

3. Lección de producción

Estar preparado para producción es, sobre todo, cuestión de disciplina. La integración debe cargar los recursos de forma determinista, soportar creaciones y eliminaciones repetidas, actualizarse sin fugas de memoria, gestionar datos ausentes o mal formados, evitar viajes de ida y vuelta excesivos al servidor y fallar de maneras que los desarrolladores puedan diagnosticar. También debe funcionar en el diseñador de Visual Studio cuando ese sea un requisito declarado. La resolución de problemas empieza por aislar el problema. Si el widget no funciona en JavaScript puro, todavía no es un problema de Wisej.NET. Si funciona en JavaScript puro pero falla dentro de Wisej.NET, inspecciona el contenedor, el orden de los scripts, la sincronización, el contexto y el flujo de llamadas al servidor. Usa `debugger`, `sourceURL`, la consola del navegador y el modelo de objetos del cliente. En el servidor, usa la depuración normal de Visual Studio y recuerda que los ensamblados cargados por el diseñador pueden quedar en caché. El proyecto final pide a los estudiantes integrar una nueva familia de widgets de JavaScript que no sea Kendo

ni DevExtreme. Kendo y DevExtreme fueron los ejemplos resueltos a lo largo del curso. El proyecto final demuestra que los mismos patrones de ciclo de vida, eventos y carga de datos se trasladan a una biblioteca desconocida, que es la verdadera prueba de que el enfoque de integración se ha entendido y no solo memorizado.

Resolver problemas en el lado del cliente

Los problemas del lado del cliente se diagnostican acotando dónde ocurren. El primer paso es la propia demostración del proveedor: si el widget falla ahí con las mismas opciones, el problema es del proveedor y no interviene código de Wisej.NET. Si la demostración funciona, lo siguiente es el InitScript inyectado, y un comentario `sourceURL` le da un nombre en las herramientas de desarrollo para poder poner puntos de interrupción en líneas reales. Un punto de interrupción en `init` muestra si existe el contenedor, si se han cargado los paquetes y qué contienen las opciones.

La consola cuenta el resto. Un error de símbolo no definido durante la carga indica un problema de orden de los paquetes. Un error de método no encontrado dentro de un callback indica un `this` perdido. Una entrada de red con un 404 indica una ruta de recurso que funcionaba en desarrollo y no en el servidor. Una cláusula de guarda que lanza un mensaje claro cuando falta una opción obligatoria convierte un widget en blanco y silencioso en un diagnóstico de una línea.

Anota cada diagnóstico en las notas de resolución de problemas con su síntoma, su causa y su solución. Las notas del proyecto final son el comienzo de un manual del equipo.

Resolver problemas en el lado del servidor

Los problemas del servidor se abordan con las herramientas normales de Visual Studio, con dos hábitos específicos de Wisej.NET. Primero, configura las excepciones para que la ejecución se detenga al lanzarse las excepciones que genera la integración, porque una excepción dentro de un manejador `WebRequest` o `WidgetEvent` podría ser capturada por el framework y manifestarse solo como un resultado ausente en el navegador. Segundo, recuerda que el Designer ejecuta el código de servidor del control en modo de diseño, así que un bloqueo ahí es un bloqueo del servidor y puede depurarse asociándose al proceso del Designer.

El Designer también guarda en caché los ensamblados cargados. Tras cambiar el código de un control, el Designer puede seguir renderizando la versión antigua hasta que se reinicie Visual Studio, lo que parece un cambio sin efecto. Reiniciar forma parte de la disciplina, no es un apaño. El código en modo de diseño debería detectar ese modo y renderizar datos de ejemplo en lugar de llamar a servicios, para que el Designer nunca necesite una base de datos.

Cuando el manejador del servidor es correcto y el navegador sigue sin mostrar nada, revisa el cuerpo de la respuesta en el panel de red antes de volver a tocar el servidor. La mayoría de los fallos restantes se deben a un content type o a la forma de la serialización.

Actualizaciones en segundo plano hechas con seguridad

Los widgets que muestran datos en vivo necesitan actualizaciones que se originan en el servidor sin una petición del usuario. El patrón tiene tres partes. `Application.StartTask` ejecuta el trabajo en un hilo en segundo plano conservando el contexto de la sesión, de modo que el código sigue pudiendo llegar a la página y al widget. Dentro de esa tarea, el servidor cambia opciones o emite un `Call` sobre el widget,

exactamente como lo haría un manejador de botón. Después, `Application.Update` envía los cambios pendientes al navegador, porque sin una petición en curso nada más los enviaría.

Hay que tener cuidado en tres puntos. La frecuencia de actualización debe estar acotada, ya que cada llamada es un viaje de ida y vuelta y un widget actualizado cien veces por segundo es una página lenta. La sesión puede terminar mientras se ejecuta la tarea, así que la tarea debe gestionar un widget ya eliminado sin lanzar excepciones. Y la llamada al proveedor debe ejecutarse en el cliente después del cambio de opción del que depende, algo que el procesamiento por lotes solo garantiza cuando ambos se emiten en la misma actualización.

Agrupar los cambios, envíalos una vez por cada cambio significativo y detén la tarea cuando se cierre la página.

La lista de comprobación de producción

El proyecto final incluye una lista de comprobación que un revisor puede aplicar a cualquier biblioteca de integración, y cada sección corresponde a un punto donde las demostraciones fallan en producción.

Recursos: cada ruta de paquete es una ruta de la aplicación o un recurso incrustado, las versiones están fijadas y no se requiere ninguna CDN. Versionado: se registran la versión del proveedor, la versión del wrapper y las notas de cambios incompatibles, y cada actualización tiene una página de prueba.

Seguridad: cada endpoint valida y autoriza, ningún objeto de dominio cruza la conexión y el HTML en bruto del proveedor se sanea.

Rendimiento: el primer renderizado carga una sola página de datos, los eventos se filtran en el cliente, las actualizaciones en segundo plano están acotadas y la eliminación destruye la instancia del proveedor y desasocia todos los manejadores para que una sesión larga no tenga fugas. Tiempo de diseño: el control renderiza datos de ejemplo en el Designer sin servicios y sin lanzar excepciones.

- ¿Puede crearse y eliminarse el wrapper cien veces sin que crezca la memoria?
- ¿Puede distinguirse un fallo del script del proveedor de un fallo de la infraestructura de Wisej.NET?
- ¿Tiene otro desarrollador todo lo necesario para actualizar el proveedor sin el autor original?



Regla de producción

Mantén claras la propiedad del estado en Wisej.NET y el comportamiento del widget de cliente. Documenta el contrato cliente/servidor antes de confiar en la integración en producción.



Error habitual que debes evitar

Copiar JavaScript de demostración en las pantallas sin documentar el ciclo de vida, los recursos, los contratos de eventos, la limpieza ni los límites de seguridad.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Resolver problemas del código de cliente con Chrome DevTools, nombres dinámicos de scripts, puntos de interrupción, la inspección en consola y las demostraciones del proveedor.
- ✓ Resolver problemas del código de servidor con Visual Studio, la configuración de excepciones, la depuración en modo de diseño y la disciplina de reiniciar el diseñador.
- ✓ Usar con cuidado las actualizaciones en segundo plano con `Application.StartTask`, las llamadas al cliente y `Application.Update`.
- ✓ Preparar listas de comprobación de recursos, versionado, seguridad y rendimiento para una biblioteca de integración.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el flujo de trabajo recomendado, los entregables requeridos y la autoevaluación antes de entregar.



Laboratorio práctico

Proyecto final: integra un widget de terceros que no conozcas como wrapper reutilizable de Wisej.NET, con llamadas al servidor, al menos un evento y carga de datos mediante postback o WebMethod.

Flujo de trabajo recomendado

- ✓ Abre el proyecto existente del curso de integración de Wisej.NET y ejecútalo en local.
- ✓ Identifica el componente de servidor, el widget de cliente, el archivo de recursos o el endpoint que intervienen en este módulo.
- ✓ Haz un solo cambio pequeño cada vez y pruébalo en el navegador antes de continuar.
- ✓ Mantén el JavaScript específico del proveedor dentro del adaptador o wrapper en lugar de repartirlo por las páginas.
- ✓ Actualiza la interfaz para que el estudiante vea con claridad el éxito, los errores, el progreso o los cambios de comunicación.
- ✓ Toma capturas de pantalla o notas que demuestren que la integración funciona y expliquen la decisión de diseño.

Entregables requeridos

- ✓ Implementación del proyecto final
- ✓ ADR de integración
- ✓ Lista de comprobación de recursos y seguridad
- ✓ Guion de demostración y notas de resolución de problemas

Autoevaluación antes de entregar

- ✓ ¿Cómo demuestras que un fallo se debe al uso de la biblioteca del proveedor y no a la infraestructura de Wisej.NET?
- ✓ ¿Qué debe limpiarse cuando se hace dispose de un widget?
- ✓ ¿Qué pertenece a una lista de comprobación de integración para producción?