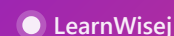


Estado, sesiones, trabajo en segundo plano y errores

Estado, sesiones y trabajo en segundo plano · Gestión segura de errores y registro



Estado, sesiones y trabajo en segundo plano

Estado de sesión por usuario frente a estado compartido (y la trampa de los campos static), además de cómo ejecutar trabajo largo en una tarea en segundo plano que no bloquea la interfaz.



Objetivo de la lección

Entiende el **estado de sesión por usuario** (y en qué se diferencia del estado compartido) y después ejecuta trabajo largo en una **tarea en segundo plano** que mantiene la página con capacidad de respuesta y actualiza el progreso de forma intencionada.

1. Estado de sesión por usuario

Una aplicación Wisej.NET se ejecuta en el servidor, pero cada usuario conectado debe sentir que tiene su propia copia de la aplicación. El **estado de sesión** son los datos que pertenecen a la sesión de un usuario conectado: el ticket seleccionado, el filtro actual, la tarea en segundo plano en curso y los mensajes de progreso de ese usuario.

Tipo de estado	Qué significa	Ejemplo para principiantes
Estado de la interfaz	Información que solo se usa para mostrar la pantalla actual.	Fila seleccionada, panel visible, texto de la etiqueta de estado.
Estado de sesión	Información que pertenece a una sesión de usuario.	La importación en curso y las tareas completadas de este usuario conectado.
Estado de negocio	Datos reales de la aplicación que deben guardarse o compartirse de forma intencionada.	Tickets almacenados por un servicio / repositorio.
Estado static / compartido	Estado compartido entre usuarios salvo que se diseñe con cuidado.	Una lista static de tareas que cualquier usuario puede modificar por accidente.



Atención: estado static / compartido

No guardes información normal por usuario en campos **static** salvo que quieras compartirla intencionadamente. Una lista static puede permitir que un usuario vea los datos o el progreso de las tareas de otro: uno de los errores de principiante más fáciles de cometer en aplicaciones de servidor.

2. Trabajo en segundo plano y páginas con capacidad de respuesta

Algunos trabajos tardan demasiado para ejecutarse directamente dentro de un clic de botón: exportar registros, importar archivos, enviar notificaciones, procesar informes. Una buena aplicación inicia la tarea, deshabilita los botones inseguros, muestra el progreso y permite que la página siga siendo utilizable mientras la tarea continúa.

```
El usuario abre la página → una sesión nueva ve su propio estado
Clic en botón           → el evento de la UI inicia la lógica del servidor
Tarea en segundo plano  → el trabajo largo se ejecuta sin bloquear la UI
Actualización del progreso → estado, barra y registro se actualizan de forma segura
Fin / error              → el usuario ve un mensaje final claro
```

3. Patrón incorrecto frente a patrón mejor

Patrón incorrecto	Patrón mejor
Poner todo el trabajo directamente en el evento de clic y bloquear la página.	Iniciar una tarea en segundo plano y actualizar el progreso y el estado de forma intencionada.
Dejar que los usuarios pulsen Start muchas veces y creen tareas duplicadas.	Deshabilitar Start mientras se ejecuta una tarea; habilitar Cancel cuando corresponda.
Mostrar solo un error genérico o dejar que la aplicación falle.	Registrar los detalles para los desarrolladores, pero mostrar al usuario un mensaje breve y amable.



Ya estás listo para la siguiente lectura

Sabes distinguir el estado de sesión del estado compartido, detectar la trampa de los campos static y describir un flujo de tarea en segundo plano con capacidad de respuesta. A continuación: gestionar los errores de forma segura y registrar el detalle útil.

Gestión segura de errores y registro

Captura excepciones y muestra mensajes seguros, registra el detalle útil y mantén legible el manejador de la tarea con try/catch/finally.



Objetivo de la lección

Captura excepciones y muestra mensajes útiles **sin** exponer detalles internos, añade un registro sencillo y mantén legible el manejador de eventos de la tarea en segundo plano con un patrón try / catch / finally claro.

1. Gestión segura de errores

La gestión de errores debe proteger tanto al usuario como al sistema. El usuario necesita un mensaje claro sobre lo que ha pasado y qué hacer a continuación; el desarrollador necesita suficiente detalle en el registro para diagnosticar. Nunca expongas cadenas de conexión, rutas de archivos, trazas de pila, secretos ni detalles internos del servidor en el mensaje para el usuario.

Detalle del error	¿Mostrarlo al usuario?	Por qué
«Import failed. Please try again or contact support.»	Sí	Útil y seguro.
<code>NullPointerException</code> en TicketService.cs, línea 52	No	Detalle interno de implementación: ponlo en un registro.
Contraseña de la base de datos o cadena de conexión	Nunca	Información sensible.
ID de la tarea y marca de tiempo	A veces	Puede ayudar a soporte a encontrar el registro, siempre que no exponga secretos.

2. Una lista de comprobación sencilla para el registro

- Registra cuándo empieza la tarea, cada paso importante, la finalización, la cancelación y el fallo.
- Incluye marcas de tiempo y nombres de tarea para que el registro sea útil más adelante.
- Mantén breves los mensajes para el usuario; deja los detalles técnicos en el registro.
- No registres contraseñas, claves de API, números de tarjeta completos ni datos privados sin un motivo sólido y sin protección.
- Al diagnosticar, comprueba: el estado de los botones, el estado de la tarea actual, el valor del progreso, el mensaje de la excepción y la entrada más reciente del registro.

3. El patrón para principiantes que debes recordar

El clic del botón inicia el flujo. La tarea en segundo plano hace el trabajo largo. La interfaz actualiza el progreso y el estado. El servicio o helper contiene la lógica reutilizable. El registro guarda los detalles. El

usuario ve mensajes seguros.

```
private async void btnStartImport_Click(object sender, EventArgs e)
{
    try
    {
        SetJobRunning(true);
        await RunImportJobAsync();
        lblStatus.Text = "Import completed successfully.";
    }
    catch (Exception ex)
    {
        LogError(ex);
        lblStatus.Text = "Import failed. Please try again or contact support.";
    }
    finally
    {
        SetJobRunning(false);
    }
}
```



Por qué importa finally

finally se ejecuta tanto si la tarea tiene éxito como si lanza una excepción, así que es el lugar adecuado para devolver los botones y el estado de la interfaz a una condición segura: la página nunca se queda bloqueada en modo «en ejecución».



Ya estás listo para el laboratorio

A continuación: mira el recorrido guiado del trabajo en segundo plano, supera la evaluación de conocimientos y después construye un ejecutor de tareas: botones Start/Cancel, una barra de progreso y un registro, un try/catch/finally que registra el detalle pero muestra un mensaje seguro y un estado de la interfaz que siempre se restablece.