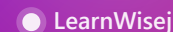


Proyecto final: aplicación Mini Helpdesk

El proyecto final · Código legible y revisión entre compañeros



El proyecto final

Qué debe incluir el mini helpdesk del proyecto final (navegación, CRUD, diálogos, validación, acabado del tema) y cómo encajan sus piezas.



Objetivo de la lección

Entiende el proyecto final: una pequeña **aplicación de helpdesk y tickets** en Wisej.NET que reúne todo lo visto en el curso: navegación, CRUD, diálogos, validación, acabado del tema, notas de arquitectura y una lista de comprobación de despliegue.

1. Qué debe demostrar el proyecto final

- La aplicación tiene un shell claro con navegación a la izquierda y pantallas separadas.
- Los tickets se pueden ver, crear, editar y eliminar mediante un `DataGridView` y diálogos.
- La validación está cerca de la interfaz, pero las reglas de negocio reutilizables están fuera de los eventos de botón sueltos.
- La aplicación tiene un tema coherente, un espaciado claro, etiquetas legibles y una ubicación predecible de los comandos.
- El proyecto incluye notas de arquitectura, una lista de comprobación para la revisión de código entre compañeros, próximos pasos y notas de despliegue.

2. Lista de comprobación de funcionalidades del proyecto final

Área funcional	Evidencia requerida
Shell de navegación	Dashboard, Tickets, Architecture, Code Review, Deployment y Next Steps se abren desde una única navegación coherente a la izquierda.
CRUD de tickets	Crear, editar y eliminar registros de tickets; un DataGridView se actualiza tras los cambios.
Diálogos y validación	Los diálogos Create/Edit Ticket validan los campos obligatorios y muestran una respuesta clara de éxito o error.
Capa de servicio	<code>TicketService</code> gestiona las operaciones de tickets en lugar de poner toda la lógica en los eventos de la interfaz.
Acabado de la interfaz	Tema aplicado, espaciado ordenado, etiquetas legibles, botones agrupados de forma coherente.
Preparación para el despliegue	La lista de comprobación de publicación cubre la configuración, la seguridad, el registro, los temas y recursos estáticos y las notas del destino.

3. Arquitectura y flujo de datos

El proyecto final une toda la aplicación. Debes poder explicar cómo circulan los datos por ella de principio a fin.

```
Program.cs    → inicia Window1
Window1.cs    → shell, navegación, páginas
Ticket.cs     → modelo de datos del ticket
TicketDialog  → formulario de creación/edición + validación
TicketService → lógica CRUD y un repositorio simulado
DataGridView  → muestra los tickets
Tema          → interfaz pulida
Despliegue    → lista de comprobación de publicación
```

El flujo: el usuario hace clic en Create/Edit, se abre el diálogo, se ejecuta la validación, `TicketService` actualiza la lista de tickets, el `DataGridView` se actualiza y el estado o las notas de despliegue registran lo que ha cambiado.



Ya estás listo para la siguiente lectura

Sabes qué debe incluir el proyecto final y cómo encajan sus piezas. A continuación: los hábitos de desarrollo que lo mantienen legible y los aspectos de la revisión entre compañeros con los que se evalúa.

Código legible y revisión entre compañeros

Los hábitos de desarrollo que mantienen legible el proyecto final y los aspectos de la revisión de código entre compañeros con los que se evalúa tu proyecto.



Objetivo de la lección

Termina el proyecto final con hábitos de desarrollador junior que hacen mantenible una aplicación Wisej.NET, y conoce lo que buscará una revisión de código entre compañeros antes de entregarlo.

1. Mantén legible el proyecto final

El proyecto final no consiste solo en que la pantalla funcione: debe mostrar hábitos que mantengan la aplicación fácil de mantener.

Hábito	Cómo se ve en el proyecto final
Nombres de control claros	Nombres como <code>btnCreateTicket</code> , <code>dgvTickets</code> , <code>lblStatus</code> , <code>txtTitle</code> , <code>cmbPriority</code> , no <code>button1</code> ni <code>label2</code> .
Manejadores de eventos pequeños	Los clics de los botones llaman a métodos auxiliares o de servicio; no esconden cientos de líneas de lógica de negocio.
Validación reutilizable	Un <code>TicketValidator</code> o helper se puede probar y reutilizar en los flujos de creación y edición.
Actualización intencionada	Después de guardar o eliminar, actualiza deliberadamente la cuadrícula y el mensaje de estado, sin efectos secundarios confusos.
Errores seguros	Muestra mensajes amables al usuario; deja los detalles internos en los registros o notas para desarrolladores.

2. En qué se centra la revisión de código entre compañeros

- **Nombres:** ¿las clases, los controles y los métodos describen claramente su función?
- **Separación de responsabilidades:** ¿la lógica de tickets está en un servicio o validador en lugar de dispersa por los eventos de la interfaz?
- **Navegación:** ¿el revisor puede seguir cómo se abre cada página?
- **Usabilidad:** ¿los botones están donde los usuarios los esperan, con etiquetas y respuestas claras?
- **Despliegue:** ¿se han comprobado la configuración, el registro, los secretos, los temas y las notas de publicación antes de la revisión?



Demuestra todo el curso

El proyecto final debe demostrar todo lo que has aprendido (navegación, CRUD, diálogos, validación, temas y despliegue), no solo una funcionalidad aislada.

3. Entrega final sugerida

Entrega una carpeta de proyecto que funcione o un repositorio de GitHub, junto con capturas de pantalla de las pantallas principales y una breve nota sobre el despliegue y la preparación. La pantalla Next Steps debe describir lo que aún necesitaría una versión lista para producción.



Ya estás listo para el laboratorio

A continuación: mira el recorrido guiado del proyecto final, supera la evaluación de conocimientos final y después construye el mini helpdesk: un shell de navegación, CRUD de tickets con diálogos y validación a través de un servicio, un tema pulido y notas de arquitectura, revisión y despliegue.