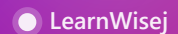


Capstone — applicazione Mini Helpdesk

Il progetto capstone · Codice leggibile e revisione tra pari



Il progetto capstone

Che cosa deve includere il mini helpdesk del capstone — navigazione, CRUD, finestre di dialogo, validazione, rifinitura del tema — e come si incastrano i suoi pezzi.



Obiettivo della lezione

Capisci il capstone: una piccola **app di helpdesk/ticket** Wisej.NET che unisce tutto ciò che hai visto nel corso — navigazione, CRUD, finestre di dialogo, validazione, rifinitura del tema, note di architettura e una checklist di rilascio.

1. Che cosa deve dimostrare il capstone

- L'app ha uno shell chiaro con navigazione a sinistra e schermate separate.
- I ticket si possono visualizzare, creare, modificare ed eliminare usando una `DataGridView` e finestre di dialogo.
- La validazione è vicina all'interfaccia, ma le regole di business riutilizzabili stanno fuori da eventi dei pulsanti sparsi.
- L'app ha un tema coerente, spaziature chiare, etichette leggibili e una posizione prevedibile dei comandi.
- Il progetto include note di architettura, una checklist per la revisione del codice tra pari, i prossimi passi e note di rilascio.

2. Checklist delle funzionalità del capstone

Area funzionale	Evidenza richiesta
Shell di navigazione	Dashboard, Tickets, Architecture, Code Review, Deployment e Next Steps si aprono da un'unica navigazione a sinistra coerente.
CRUD dei ticket	Creazione, modifica ed eliminazione dei record dei ticket; una DataGridView si aggiorna dopo le modifiche.
Finestre di dialogo e validazione	Le finestre Create/Edit Ticket validano i campi obbligatori e mostrano un feedback chiaro di successo/errore.
Livello di servizio	<code>TicketService</code> gestisce le operazioni sui ticket invece di mettere tutta la logica negli eventi dell'interfaccia.
Rifinitura dell'interfaccia	Tema applicato, spaziature sistemate, etichette leggibili, pulsanti raggruppati in modo coerente.
Prontezza al rilascio	La checklist di rilascio copre configurazione, sicurezza, logging, risorse di tema/statiche e note sulla destinazione.

3. Architettura e flusso dei dati

Il capstone tiene insieme l'intera app. Dovresti saper spiegare come i dati la attraversano dall'inizio alla fine.

```
Program.cs    → avvia Window1
Window1.cs    → shell, navigazione, pagine
Ticket.cs     → modello dati del ticket
TicketDialog  → form di creazione/modifica + validazione
TicketService → logica CRUD e un repository fittizio
DataGridView  → mostra i ticket
Tema          → interfaccia rifinita
Rilascio      → checklist di rilascio
```

Il flusso: l'utente fa clic su Create/Edit, la finestra di dialogo si apre, la validazione viene eseguita, `TicketService` aggiorna l'elenco dei ticket, la `DataGridView` si aggiorna, e lo stato o le note di rilascio registrano che cosa è cambiato.



Sei pronto per la prossima lettura

Sai che cosa deve includere il capstone e come si incastrano i suoi pezzi. Prossimo passo: le abitudini da sviluppatore che lo mantengono leggibile, e i punti della revisione tra pari su cui viene valutato.

Codice leggibile e revisione tra pari

Le abitudini da sviluppatore che mantengono leggibile il capstone, e i punti della revisione del codice tra pari su cui viene valutato il tuo progetto.



Obiettivo della lezione

Completa il capstone con le abitudini da sviluppatore junior che rendono manutenibile un'app Wisej.NET, e sappi che cosa cercherà una revisione del codice tra pari prima della consegna.

1. Mantieni leggibile il capstone

Il progetto finale non consiste solo nel far funzionare la schermata — deve mostrare le abitudini che mantengono l'app facile da mantenere.

Abitudine	Come si vede nel capstone
Nomi chiari per i controlli	Nomi come <code>btnCreateTicket</code> , <code>dgvTickets</code> , <code>lblStatus</code> , <code>txtTitle</code> , <code>cmbPriority</code> — non <code>button1</code> o <code>label2</code> .
Gestori di eventi piccoli	I click dei pulsanti chiamano metodi helper o di servizio; non nascondono centinaia di righe di logica di business.
Validazione riutilizzabile	Un <code>TicketValidator</code> o un helper può essere testato e riutilizzato dai flussi di creazione/modifica.
Aggiornamento intenzionale	Dopo un salvataggio/un'eliminazione, aggiorna di proposito la griglia e il messaggio di stato — niente effetti collaterali che confondono.
Errori sicuri	Mostra messaggi comprensibili all'utente; tieni i dettagli interni nei log o nelle note per gli sviluppatori.

2. I punti della revisione del codice tra pari

- **Nomi:** classi, controlli e metodi descrivono chiaramente il loro compito?
- **Separazione delle responsabilità:** la logica dei ticket sta in un servizio/validator invece di essere sparsa tra gli eventi dell'interfaccia?
- **Navigazione:** chi rivede riesce a seguire come si apre ogni pagina?
- **Usabilità:** i pulsanti sono dove gli utenti se li aspettano, con etichette e feedback chiari?
- **Rilascio:** configurazione, logging, segreti, temi e note di rilascio sono stati controllati prima della revisione?



Mostra l'intero corso

Il capstone deve dimostrare tutto ciò che hai imparato — navigazione, CRUD, finestre di dialogo, validazione, temi e rilascio — non solo una funzionalità isolata.

3. Consegna finale consigliata

Consegna una cartella di progetto funzionante o un repository GitHub, più gli screenshot delle schermate principali e una breve nota sul rilascio e sulla prontezza. La schermata Next Steps deve descrivere che cosa servirebbe ancora a una versione pronta per la produzione.



Sei pronto per il lab

Prossimo passo: guarda la videoguia del capstone, supera la verifica delle conoscenze finale, poi costruisci il mini helpdesk — uno shell di navigazione, il CRUD dei ticket con finestre di dialogo e validazione tramite un servizio, un tema rifinito e note su architettura, revisione e rilascio.