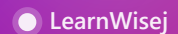


Abschlussprojekt – Mini-Helpdesk-Anwendung

Das Abschlussprojekt · Lesbarer Code und Peer-Review



Das Abschlussprojekt

Was der Mini Helpdesk des Abschlussprojekts enthalten muss – Navigation, CRUD, Dialoge, Validierung, Theme-Feinschliff – und wie seine Teile zusammenpassen.



Lernziel der Lektion

Verstehen Sie das Abschlussprojekt: eine kleine Wisej.NET-**Helpdesk-/Ticket-App**, die alles aus dem Kurs verbindet – Navigation, CRUD, Dialoge, Validierung, Theme-Feinschliff, Architektur-Notizen und eine Deployment-Checkliste.

1. Was das Abschlussprojekt zeigen soll

- Die App hat eine klare Shell mit linker Navigation und getrennten Seiten.
- Tickets lassen sich mit einer `DataGridView` und Dialogen anzeigen, anlegen, bearbeiten und löschen.
- Die Validierung liegt nah an der Oberfläche, wiederverwendbare Geschäftsregeln leben aber außerhalb beliebiger Button-Events.
- Die App hat ein einheitliches Theme, klare Abstände, gut lesbare Beschriftungen und eine vorhersehbare Anordnung der Befehle.
- Das Projekt enthält Architektur-Notizen, eine Checkliste für das Peer-Code-Review, nächste Schritte und Deployment-Notizen.

2. Funktions-Checkliste für das Abschlussprojekt

Funktionsbereich	Geforderter Nachweis
Navigationsshell	Dashboard, Tickets, Architecture, Code Review, Deployment und Next Steps öffnen sich über eine einheitliche linke Navigation.
Ticket-CRUD	Ticket-Datensätze anlegen, bearbeiten und löschen; eine DataGridView wird nach Änderungen aktualisiert.
Dialoge & Validierung	Die Dialoge zum Anlegen und Bearbeiten von Tickets validieren Pflichtfelder und geben klares Feedback zu Erfolg und Fehlern.
Service-Schicht	<code>TicketService</code> übernimmt die Ticket-Operationen, statt die gesamte Logik in UI-Events zu legen.
UI-Feinschliff	Theme angewendet, Abstände bereinigt, Beschriftungen gut lesbar, Buttons einheitlich gruppiert.
Deployment-Reife	Die Release-Checkliste deckt Konfiguration, Sicherheit, Logging, Theme/statische Ressourcen und Notizen zum Zielsystem ab.

3. Architektur und Datenfluss

Das Abschlussprojekt fügt die gesamte App zusammen. Sie sollten erklären können, wie die Daten von Anfang bis Ende durch sie fließen.

```
Program.cs      → startet Window1
Window1.cs      → Shell, Navigation, Seiten
Ticket.cs       → Datenmodell für Tickets
TicketDialog    → Formular zum Anlegen/Bearbeiten + Validierung
TicketService   → CRUD-Logik und ein Fake-Repository
DataGridView    → zeigt die Tickets an
Theme           → verfeinerte Oberfläche
Deployment      → Release-Checkliste
```

Der Ablauf: Der Benutzer klickt auf Create/Edit, der Dialog öffnet sich, die Validierung läuft, `TicketService` aktualisiert die Ticketliste, die `DataGridView` wird aktualisiert, und die Status- oder Deployment-Notizen halten fest, was sich geändert hat.



Sie sind bereit für die nächste Lektüre

Sie wissen, was das Abschlussprojekt enthalten muss und wie seine Teile zusammenpassen. Als Nächstes: die Entwickलगewohnheiten, die es lesbar halten, und die Schwerpunkte des Peer-Reviews, nach denen es bewertet wird.

Lesbarer Code und Peer-Review

Die Entwicklergewohnheiten, die das Abschlussprojekt lesbar halten, und die Schwerpunkte des Peer-Code-Reviews, nach denen Ihr Projekt bewertet wird.



Lernziel der Lektion

Schließen Sie das Abschlussprojekt mit Gewohnheiten eines Junior-Entwicklers ab, die eine Wisej.NET-App wartbar machen, und wissen Sie, worauf ein Peer-Code-Review achtet, bevor Sie abgeben.

1. Das Abschlussprojekt lesbar halten

Beim Abschlussprojekt geht es nicht nur darum, dass die Seite funktioniert – es soll Gewohnheiten zeigen, die die App leicht wartbar halten.

Gewohnheit	Wie sie im Abschlussprojekt aussieht
Eindeutige Control-Namen	Namen wie <code>btnCreateTicket</code> , <code>dgvTickets</code> , <code>lblStatus</code> , <code>txtTitle</code> , <code>cmbPriority</code> – nicht <code>button1</code> oder <code>label2</code> .
Kleine Event-Handler	Button-Klicks rufen Hilfs- oder Service-Methoden auf; sie verbergen keine Hunderte Zeilen Geschäftslogik.
Wiederverwendbare Validierung	Ein <code>TicketValidator</code> oder eine Hilfsklasse lässt sich testen und von den Abläufen zum Anlegen und Bearbeiten wiederverwenden.
Gezieltes Aktualisieren	Nach Speichern oder Löschen aktualisieren Sie Grid und Statusmeldung bewusst – ohne verwirrende Nebeneffekte.
Sichere Fehler	Zeigen Sie verständliche Meldungen für Benutzer; interne Details gehören in Entwickler-Logs oder Notizen.

2. Schwerpunkte des Peer-Code-Reviews

- **Benennung:** Beschreiben Klassen, Controls und Methoden ihre Aufgabe eindeutig?
- **Trennung der Zuständigkeiten:** Liegt die Ticketlogik in einem Service/Validator, statt über UI-Events verstreut zu sein?
- **Navigation:** Kann der Reviewer nachvollziehen, wie sich jede Seite öffnet?
- **Bedienbarkeit:** Sind Buttons dort platziert, wo Benutzer sie erwarten, mit klaren Beschriftungen und Feedback?
- **Deployment:** Wurden Konfiguration, Logging, Secrets, Themes und Release-Notizen vor dem Review geprüft?



Zeigen Sie den ganzen Kurs

Das Abschlussprojekt sollte alles zeigen, was Sie gelernt haben – Navigation, CRUD, Dialoge, Validierung, Theming und Deployment –, nicht nur eine einzelne, isolierte Funktion.

3. Empfohlene Abgabe

Geben Sie einen funktionierenden Projektordner oder ein GitHub-Repository ab, dazu Screenshots der wichtigsten Seiten und eine kurze Notiz zu Deployment und Release-Reife. Die Seite Next Steps sollte beschreiben, was eine produktivreife Version noch bräuchte.



Sie sind bereit für das Lab

Als Nächstes: Sehen Sie sich den Video-Walkthrough zum Abschlussprojekt an, bestehen Sie den abschließenden Wissenstest und erstellen Sie dann den Mini Helpdesk – eine Navigationshell, Ticket-CRUD mit Dialogen und Validierung über einen Service, ein verfeinertes Theme sowie Notizen zu Architektur, Review und Deployment.