

Dialoge, Validierung und Workflows

Dialoge und modale Workflows · Validierung und das Speichern/Abbrechen-Muster

Dialoge und modale Workflows

Was ein Dialog ist, wann Sie ihn statt einer Seite verwenden, der modale Workflow in fünf Schritten und wo Meldungsfenster helfen.



Lernziel

Verstehen Sie **Dialoge** und den modalen Workflow zum Anlegen und Bearbeiten, auf den Geschäftsanwendungen setzen: ein fokussiertes Fenster öffnen, den Benutzer eine Aufgabe erledigen lassen, riskante Aktionen bestätigen lassen und ein eindeutiges Ergebnis zurückgeben.

1. Was ist ein Dialog?

Ein **Dialog** ist eine kleinere Maske, die über der Hauptseite erscheint, damit der Benutzer eine fokussierte Aufgabe erledigen kann. In diesem Kurs sind die häufigsten Dialoge **Create Ticket** und **Edit Ticket**. Ein Dialog sollte nicht die ganze Anwendung sein wollen – er sammelt die benötigten Werte, validiert sie und gibt ein eindeutiges Ergebnis zurück.



Kernidee

Ein Dialog ist *modal*: Der Benutzer schließt ihn ab oder bricht ihn ab, bevor er zur Seite dahinter zurückkehrt. Genau dieser Fokus macht Workflows zum Anlegen und Bearbeiten vorhersehbar.

2. Dialog oder Seite

Art der Maske	Beste Verwendung	Beispiel
Seite / Hauptinhaltsbereich	Große App-Masken, in denen Benutzer Zeit verbringen und zwischen Funktionen navigieren.	Tickets-Seite, Customers-Seite, Settings-Seite.
Dialog / modales Fenster	Ein kleiner, fokussierter Workflow, der abgeschlossen oder abgebrochen werden soll, bevor es zurückgeht.	Dialog Create Ticket, Dialog Edit Ticket, Meldung Confirm Delete.

3. Das gängige Muster für modale Workflows

Fast jede Funktion zum Anlegen oder Bearbeiten folgt demselben Ablauf in fünf Schritten. Wer sich daran hält, erhält einen lesbaren und vorhersehbaren Workflow.

Button-Klick	→ Benutzer wählt New oder Edit
Dialog öffnen	→ modales Formular / Fenster anzeigen
Validieren	→ Pflichtfelder prüfen
Speichern	→ Service-Schicht aufrufen
UI aktualisieren	→ Grid aktualisieren und Rückmeldung zeigen

Der Benutzer klickt auf **New** oder **Edit**, die App öffnet einen Dialog, der Dialog validiert die Felder, der Button **Save** ruft einen Service auf, und die Hauptseite wird aktualisiert, *nachdem* der Dialog erfolgreich geschlossen wurde.

4. Meldungsfenster und Bestätigungen

Verwenden Sie Meldungsfenster, wenn der Benutzer sofortige Rückmeldung braucht oder eine riskante Aktion bestätigen muss.

Situation	Empfohlene Rückmeldung
Speichern abgeschlossen	Eine kurze Erfolgsmeldung anzeigen und die Ticketliste aktualisieren.
Pflichtfeld fehlt	Eine klare Validierungsmeldung neben dem Feld oder oben im Dialog anzeigen.
Ausgewähltes Ticket löschen	Vor dem Löschen eine Bestätigung einholen.
Abbrechen nach geänderten Werten	Optional: darauf hinweisen, dass ungespeicherte Änderungen verloren gehen.



Sie sind bereit für die nächste Lektüre

Sie können beschreiben, was ein Dialog ist, wann ein Dialog statt einer Seite sinnvoll ist, wie der modale Workflow in fünf Schritten abläuft und wo Meldungsfenster helfen. Als Nächstes: Eingaben validieren und das einheitliche Speichern/Abbrechen-Muster.

Validierung und das Speichern/Abbrechen-Muster

Validieren Sie nah an der UI, ohne Geschäftsregeln dort einzusperren, schreiben Sie einen lesbaren Save-Handler und wenden Sie ein einheitliches Speichern/Abbrechen-Muster an.



Lernziel

Validieren Sie Eingaben nah an der UI, ohne Geschäftsregeln dort einzusperren, schreiben Sie einen lesbaren Save-Handler und wenden Sie ein einheitliches Speichern/Abbrechen-Muster an, damit Benutzer immer wissen, was passiert ist.

1. Validierung: nah an der UI, aber nicht in ihr gefangen

Die Validierung gehört so nah an die UI, dass Benutzer sofort Rückmeldung bekommen – wichtige Geschäftsregeln sollten sich aber trotzdem außerhalb des Button-Klicks testen lassen. Der Dialog prüft einfache Feldprobleme und ruft für wiederverwendbare Regeln eine Service-Klasse auf.

Art der Validierung	Beispiel	Geeigneter Ort
Einfache UI-Validierung	Das Titel-Textfeld ist leer; es ist keine Priorität ausgewählt.	Eine Dialogmethode wie <code>ValidateForm()</code> .
Geschäftsregel	Ein geschlossenes Ticket darf von einem Support Agent nicht bearbeitet werden.	<code>TicketService</code> oder ein Validierungs-Service.
Persistenzregel	Die Ticket-ID existiert bereits im Speicher.	Die Repository- bzw. Service-Schicht.

2. Sicheres Event-Handling

Button-Klick-Events sollten lesbar sein. Verstecken Sie langsame Datenbankaufrufe oder komplizierte Geschäftslogik nicht in einem banalen UI-Event, außer es ist beabsichtigt und klar benannt. Ein gutes Save-Event ruft kleine Hilfsmethoden wie `ValidateForm()`, `BuildTicketFromFields()` und `ticketService.SaveTicket(ticket)` auf.

```
private void btnSave_Click(object sender, EventArgs e)
{
    if (!ValidateForm())
        return;

    Ticket ticket = BuildTicketFromFields();
    ticketService.SaveTicket(ticket);

    this.DialogResult = DialogResult.OK;
    this.Close();
}
```



Erst validieren, dann speichern

Der Handler kehrt frühzeitig zurück, wenn `ValidateForm()` fehlschlägt – nichts wird gespeichert und der Dialog bleibt offen. Nur gültige Eingaben erreichen den Service.

3. Ein einheitliches Speichern/Abbrechen-Muster

- **Save** validiert zuerst, ruft dann den Service auf und schließt den Dialog *erst nach erfolgreichem Speichern*.
- **Cancel** schließt den Dialog, ohne zu speichern.
- Fehlermeldungen sollten dem Benutzer genau sagen, was er korrigieren muss.
- Die übergeordnete Seite aktualisiert das Grid nur, wenn der Dialog **OK** zurückgibt.
- Benennen Sie Buttons eindeutig: `btnSave`, `btnCancel`, `btnNew`, `btnEdit`.

4. Entwicklungsgewohnheiten für dieses Modul

- Benennen Sie Controls eindeutig, damit der Event-Code lesbar bleibt.
- Halten Sie das Dialog-Layout einfach und fokussiert.
- Trennen Sie UI-Zustand, Geschäftsregeln und gespeicherte Daten.
- Nutzen Sie Services für wiederverwendbare Ticket-Operationen.
- Setzen Sie Meldungsfenster und Validierungsmeldungen einheitlich ein.



Sie sind bereit für das Lab

Als Nächstes: Sehen Sie sich den Video-Walkthrough zum Dialog an, bestehen Sie den Wissenstest und bauen Sie dann wiederverwendbare Dialoge zum Anlegen und Bearbeiten von Tickets – erst validieren, über den Service speichern, `DialogResult.OK` zurückgeben und das Grid mit klarer Rückmeldung aktualisieren.