Projektdokumentation – Tool2Go

Konsolenbasierte Anwendung zur Verwaltung eines Werkzeugverleihs

---

Entwicklungsumgebung

- **IDE**: Microsoft Visual Studio Community 2022
- **Version**: 17.11.5
- **Framework**: .NET 8.0
- **Sprache**: C#
- **Projektart**: Konsolenanwendung
- **Build-Verzeichnis**: bin/Debug/net8.0
- **Startdatei**: Tool2Go.exe
- **Beispieldaten**: kunden.xml, kategorien.xml, buchungen.xml

Beim Start werden automatisch alle XML-Dateien geladen, um das System mit Beispieldaten zu initialisieren. Die .exe-Datei befindet sich im bin/Debug-Ordner und kann ohne Kompilierung ausgeführt werden.

Projektüberblick

Tool2Go ist eine textbasierte Verwaltungsanwendung zur Abwicklung von Werkzeugbuchungen durch Kunden. Die Anwendung ermöglicht:

- Kundenverwaltung
- Verwaltung von Werkzeugkategorien und Werkzeugen
- Buchung mehrerer Werkzeugtypen mit Einzel- oder Globalzeitraum
- Alters- & Versicherungsprüfung bei Buchung
- Datenpersistenz durch XML-Dateien
- Rückgängig-Option bei versehentlichen Eingaben

Getroffene Annahmen & Designentscheidungen

1. Mehrere Werkzeuginstanzen

Werkzeuge werden **nicht mit Mengenangabe**, sondern als **Einzelinstanzen** abgebildet. Für 4 identische Werkzeuge vom selben Typ werden 4 Objekte erstellt. Vorteil: präzise Verwaltung, Buchung und Konfliktvermeidung.

2. Typdefinition

Ein Werkzeugtyp wird durch die Kombination **Hersteller + Modell** definiert. Werkzeuge mit gleichem Typ, aber unterschiedlichen technischen Daten, gelten als **eigene Typen**.

3. Buchungszeiträume

- Der Nutzer kann wählen, ob ein **globaler Zeitraum** für alle Werkzeuge gelten soll.
- Bei **individuellen Zeiträumen**:
    - Das globale Start-/Enddatum wird beim ersten Werkzeug **automatisch übernommen**.
    - Für alle weiteren erfolgt eine eigene Eingabe.

4. Rücksprungoption bei versehentlichen Eingaben

Während des Hinzufügens weiterer Werkzeuge kann der Nutzer jederzeit den Befehl **„zurück“** eingeben, um **den aktuellen Schritt abzubrechen** und zurück zum Prompt „Noch ein weiteres Werkzeug buchen?“ zu gelangen. Die Buchung bleibt dabei erhalten.

5. Temporäre Verfügbarkeitsprüfung

Während einer laufenden Buchung werden bereits ausgewählte Werkzeuge **temporär bei der Verfügbarkeit berücksichtigt**, um Doppelbuchungen innerhalb derselben Buchung zu vermeiden.

Probleme & Lösungen im Entwicklungsprozess

Werkzeuge mit int-Anzahl statt Instanzobjekten

Anfangs wurde die Anzahl identischer Werkzeuge mit einem int-Wert gespeichert. Dies führte zu Problemen bei Buchung, Verfügbarkeit und Gruppierung. Lösung: Jedes Werkzeug wird als **eigene Instanz** erstellt.

Überbuchung trotz Begrenzung

Werkzeuge konnten mehrfach in derselben Buchung gebucht werden, weil temporär reservierte Werkzeuge nicht berücksichtigt wurden. Lösung: AnzahlFreierInstanzenVomTyp(...) wurde so angepasst, dass auch **temporäre Buchungspositionen** geprüft werden.

Unbeabsichtigte Endlosschleifen

Nach „j“ auf die Frage „Noch ein weiteres Werkzeug buchen?“ wurde ein Werkzeug erzwungen – auch bei versehentlichen Klicks. Lösung: **„zurück“** bricht das aktuelle Werkzeug-Hinzufügen ab und bringt den Nutzer zurück zum Entscheidungspunkt.

Redundante Datumseingaben

Wenn kein globaler Zeitraum gewählt wurde, wurde der Nutzer **auch beim ersten Werkzeug** zur Eingabe des Datums aufgefordert. Lösung: Das initial eingegebene Datum wird **beim ersten Werkzeug automatisch übernommen**, bei den folgenden individuell abgefragt.

Löschen von Kategorien ohne Warnung

Werkzeugkategorien konnten gelöscht werden, ohne dass gewarnt wurde, dass auch alle zugehörigen Werkzeuge entfernt werden. Lösung: Warnung + Bestätigungsabfrage vor dem Löschen.

Werkzeug-Typ in mehreren Kategorien

Ein identischer Werkzeugtyp (gleicher Hersteller + Modell) konnte fälschlich in mehreren Kategorien existieren. Lösung: Vor dem Hinzufügen wird geprüft, ob der Typ in einer anderen Kategorie existiert. Falls ja, erscheint eine Fehlermeldung mit Angabe, in **welcher Kategorie** der Typ bereits vorhanden ist.

Beispielstruktur beim Start

Beim ersten Start (sofern noch keine XML-Dateien vorhanden sind) werden automatisch Beispiel-Daten generiert:

Kunden

- 1x Kunde mit IBAN & über 21 Jahren
- 1x Kunde ohne IBAN & unter 21 Jahren

Werkzeugkategorien mit Werkzeugen

Es werden vier Kategorien mit realistischen Werkzeugen erstellt:

- Manche Werkzeuge haben **mehrere Instanzen**
- Manche unterscheiden sich nur durch **technische Daten**
- Manche sind **einzigartig**

Beispiel:

- **Bagger**: 3x „Caterpillar 301.7D“ mit unterschiedlichen technischen Daten
- **Bohrhammer**: 4x „Bosch X3“, 1x „Makita BHR202“
- **Rüttelplatte**: 2x pro Typ
- **Winkelschleifer**: Mischung aus doppelten und einzelnen Typen

---

Aufbau der Persistenzschicht (XML-Dateien)

Beim Beenden werden alle Daten in folgende XML-Dateien gespeichert:

- kunden.xml: Enthält alle Kunden mit IBAN, Geburtsdatum, Adresse
- kategorien.xml: Enthält alle Werkzeugkategorien und zugehörige Werkzeuge
- buchungen.xml: Enthält Buchungen inkl. Werkzeugliste, Kunde, Start-/Enddatum, Versicherungspflicht

Beim Start des Programms werden diese Daten wieder geladen. Existieren sie nicht, werden sie automatisch generiert.

Verwendete Bibliotheken

- Es wurden **keine externen NuGet-Pakete** verwendet.
- Alle Funktionen basieren auf .NET-eigenen Namespaces:
    - System
    - System.Linq
    - System.Xml.Serialization
    - System.Globalization

Projektstruktur & Architekturentscheidungen

Service-Architektur und Interface-Ansatz

Das Projekt verwendet eine **Service-Architektur**, bei der fachliche Logik (z. B. Verwaltung von Kunden, Buchungen oder Werkzeugkategorien) in separaten Service-Klassen gekapselt wird. Dadurch wird die Trennung zwischen Benutzeroberfläche (z. B. Konsolenmenü) und Geschäftslogik gefördert. Jeder Service implementiert ein gemeinsames Interface IVerwaltbar<T>, was eine einheitliche Struktur für CRUD-Funktionalitäten bietet (z. B. Anzeigen, Hinzufügen, Löschen, Bearbeiten).

Vorteile dieses Ansatzes:

- **Hohe Wartbarkeit**: Änderungen an der Logik einzelner Entitäten (z. B. Kunden, Kategorien) erfordern keine Änderung in anderen Teilen des Codes.
- **Wiederverwendbarkeit**: Die gemeinsamen Methoden lassen sich generisch behandeln, z. B. in Menüs.
- **Austauschbarkeit**: Services könnten künftig gegen datenbankgestützte Varianten ausgetauscht werden, ohne das UI zu ändern.

Ordnerstruktur und Schichten:

- **Models/**
  Enthält alle Datenmodelle (z. B. Kunde, Werkzeug, BuchungPos, etc.). Diese Klassen repräsentieren die Struktur der Objekte und enthalten nur Daten, keine Logik.
- **Services/**
  Hier befinden sich alle Service-Klassen, die die Geschäftslogik kapseln, z. B. KundenService, BuchungService oder WerkzeugKategorieService.
- **Utils/**
  Enthält Hilfsklassen wie den InputHelper und EingabeParser, die die Eingabelogik vereinheitlichen.

EingabeParser und InputHelper

Benutzereingaben werden zentral über den InputHelper abgewickelt. Dieser verwendet definierte Parser wie EingabeParser.Int oder EingabeParser.Bool, um Eingaben zu validieren und Fehlermeldungen konsistent darzustellen.

Zusätzlich wurde ein Mechanismus eingebaut, mit dem man Eingaben durch "abbrechen" oder "zurück" abbrechen kann – etwa bei versehentlichen Aktionen. Diese führen zu einer OperationCanceledException, die kontrolliert abgefangen wird.

Vorteile:

- **Konsistenz**: Fehlerbehandlung und Validierung erfolgen an einer Stelle.
- **Wiederverwendbarkeit**: Alle Eingaben nutzen ein zentrales, einheitliches System.
- **Erweiterbarkeit**: Neue Eingabetypen können einfach ergänzt werden.