Inhalt

Beschreibung des Projekts

Ziel des Projekts war es, eine digitale Unterstützung für einen Zwei-Personen-Haushalt zu entwickeln. Im Zentrum steht eine gemeinsame To-Do-Liste, auf der alltägliche Aufgaben, wie z.B. das Bad oder die Küche putzen verwaltet werden und einer Person zugewiesen werden können.

Die Aufgabenliste sollte sowohl automatisch im festen Rhythmus befüllt werden, als auch manuell anpassbar sein - um etwa auf unvorhergesehene Ereignisse flexibel reagieren zu können.

Die Übernahme und Aufteilung der anstehenden Aufgaben erfolgt anschließend in individuellen „Teil-Listen“, die den beteiligten Personen zugeordnet sind. Dadurch bleibt die Übersicht erhalten, wer welcher Aufgaben zugewiesen ist.

Zur Motivation wurde ein Punktesystem implementiert. Es bietet einen spielerischen Ansatz, um den Haushaltsaufwand sichtbar und vergleichbar zu machen. So kann nachvollzogen werden, wer in letzter Zeit mehr Aufgaben oder besonders wertvolle übernommen hat – z. B. das Reinigen eines großen Badezimmers im Vergleich zum Entleeren eines Mülleimers.

Ziele des Projekts

- **Einfache und intuitive Benutzerführung**

- **Grafische Benutzeroberfläche (GUI)** auf Basis von **WPF**

- **Anpassbare Aufgabeninhalte**, sowohl im wiederkehrenden **Rhythmus** als auch in der **Punktevergabe**:

    - Unterschiede in Haushaltsgröße (großes Bad = mehr Punkte)
    - Individuelle Aufgabengestaltung (optional Gartenarbeit)

- Unterstütze Eingaben durch sinnvolle Vorschläge

- „**Intelligente Wiederholungen“:**
  Aufgaben sollten nicht mehrfach gleichzeitig erscheinen – Redundanz vermeiden

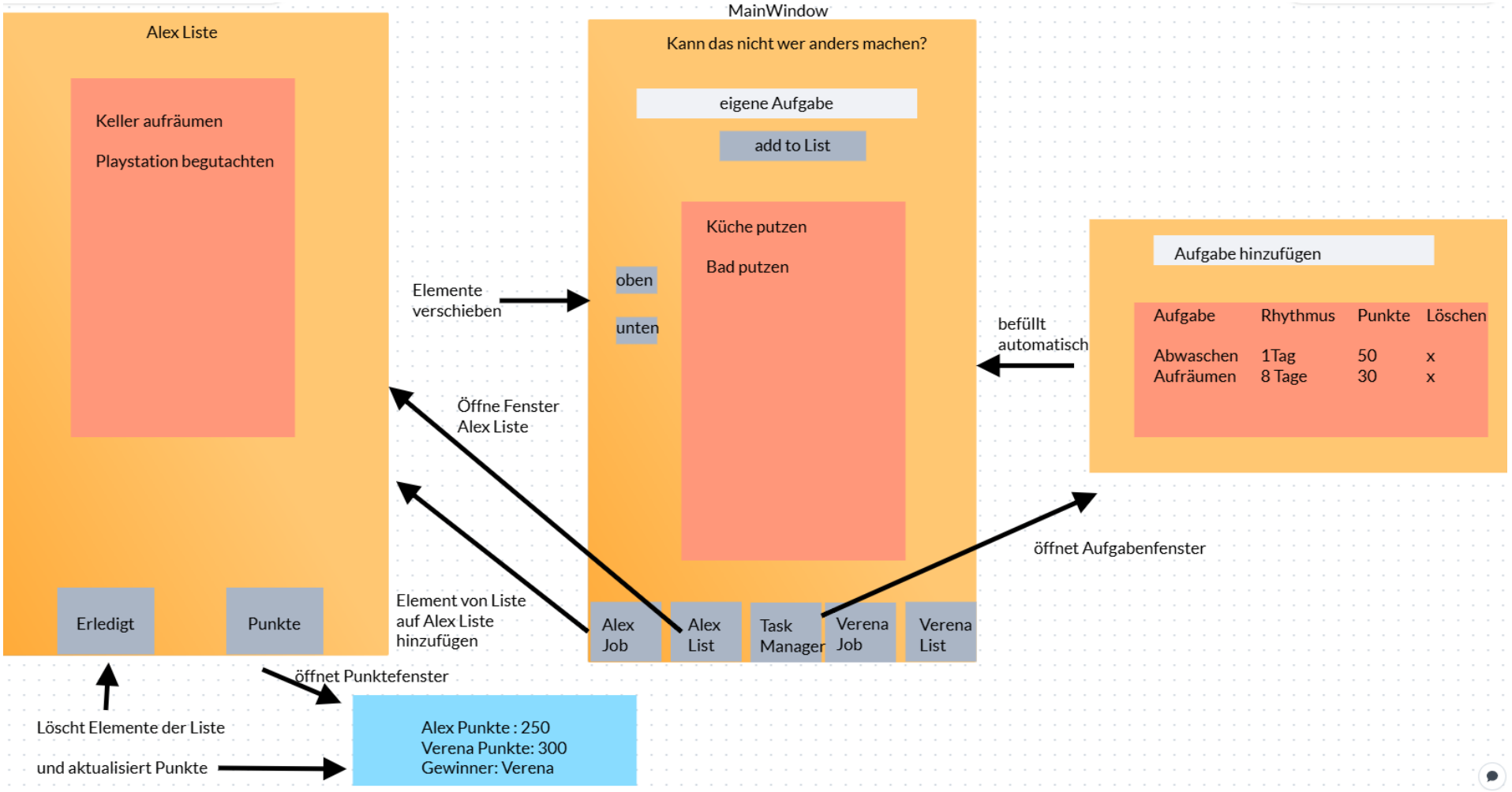
- **Spielerischer Ansatz:**
  Motivation durch Belohnung in Form von Punkten und Ernennung eines Siegers

- **Speicherung** des aktuellen Zustands beim Verlassen der App

- **Laden** des letzten Zustands beim Starten der App

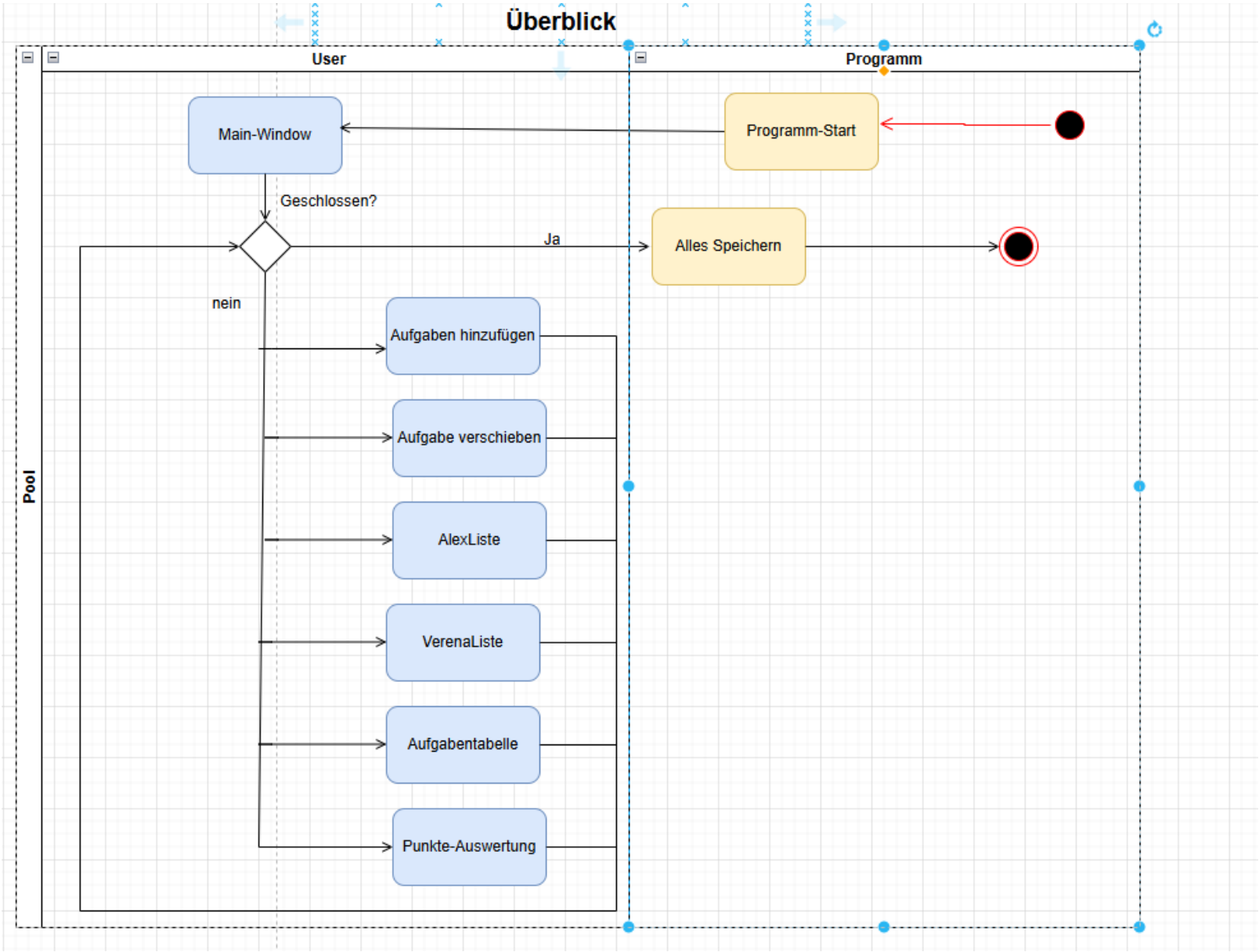
Programmlogik

Die folgende Darstellung zeigen die grundlegende Idee mithilfe eines Mockups

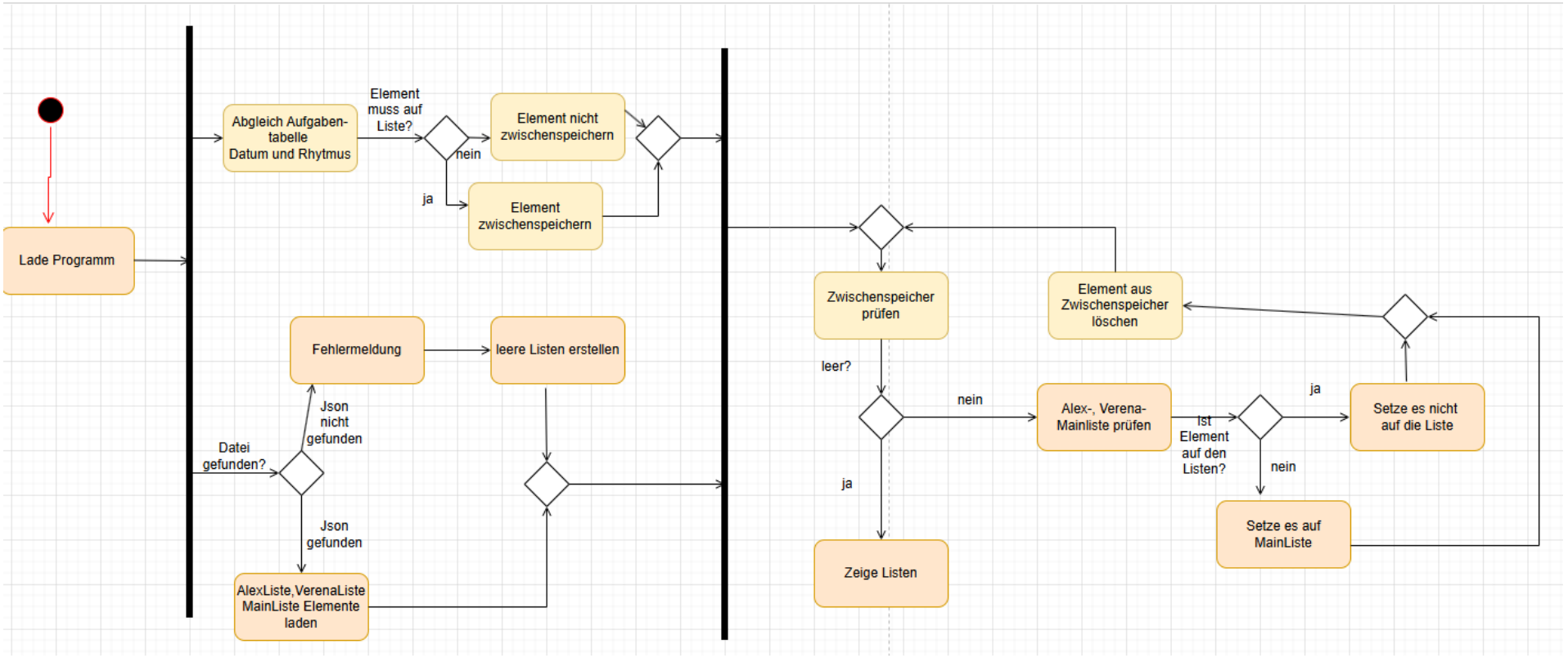


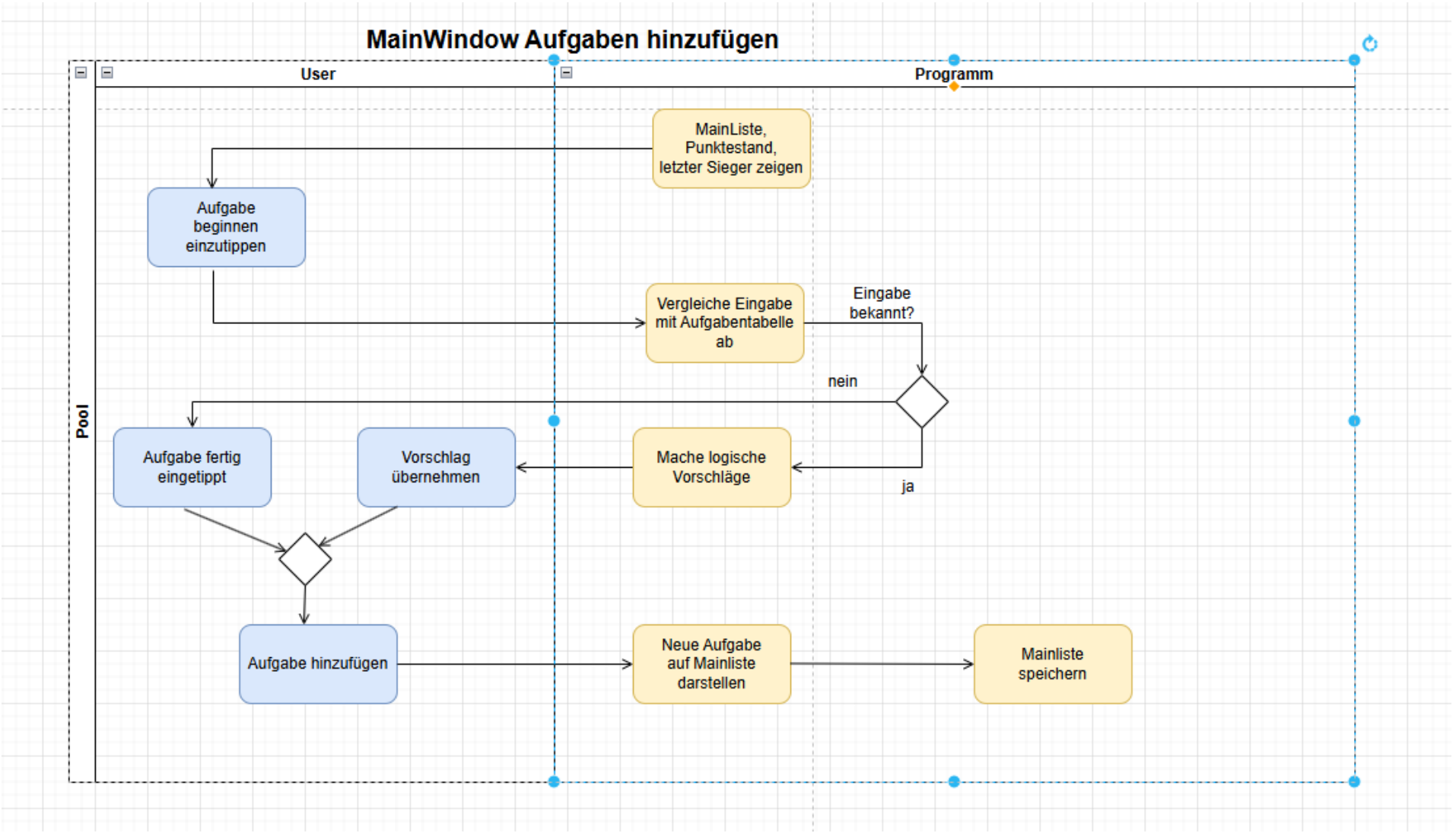
UML – Diagramme

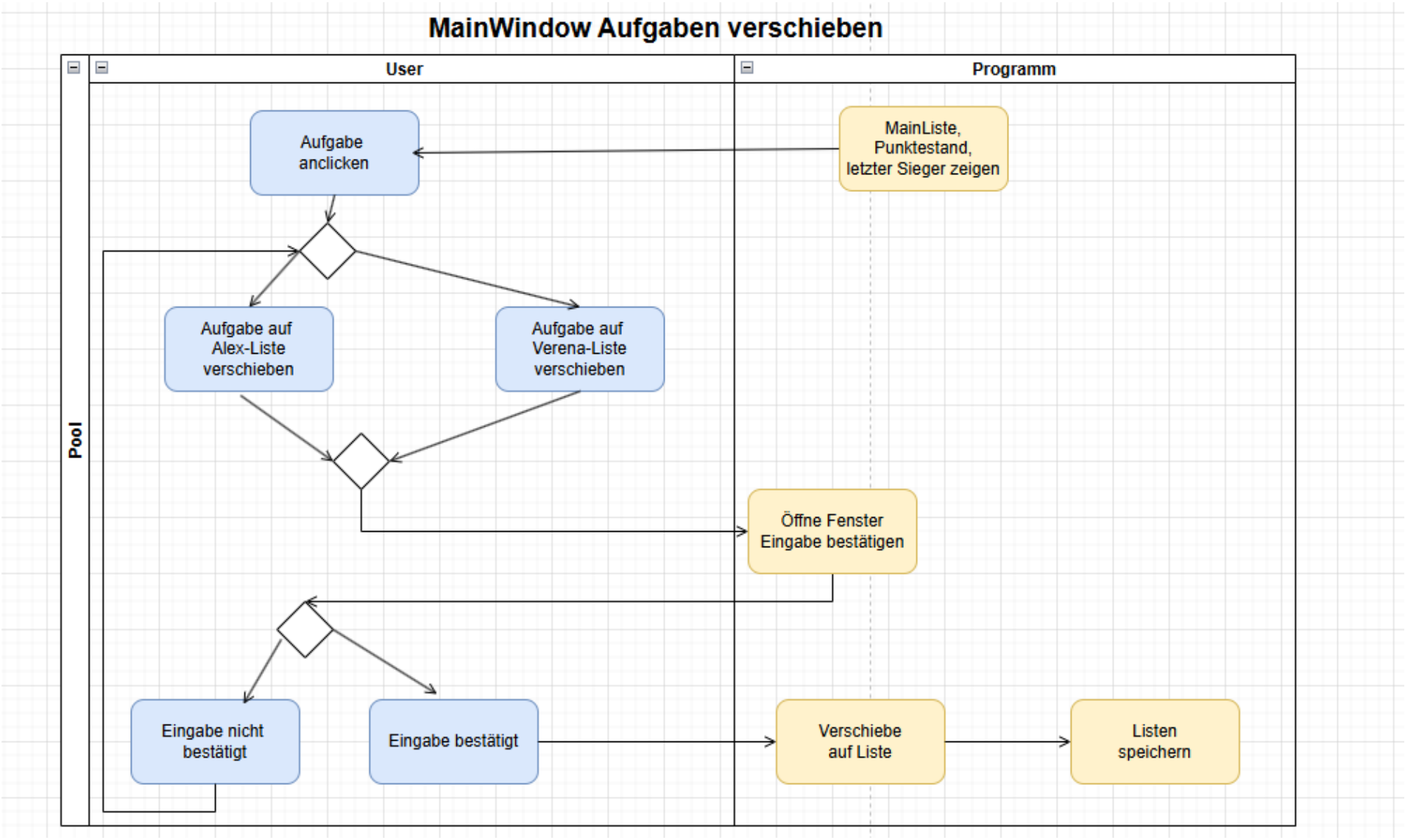
Da es nicht möglich war ein übersichtliches Aktivitätsdiagramm für alle Funktionen des Programms zu schreiben, habe ich die verschiedenen Aktivitäten einzeln aufgegliedert.

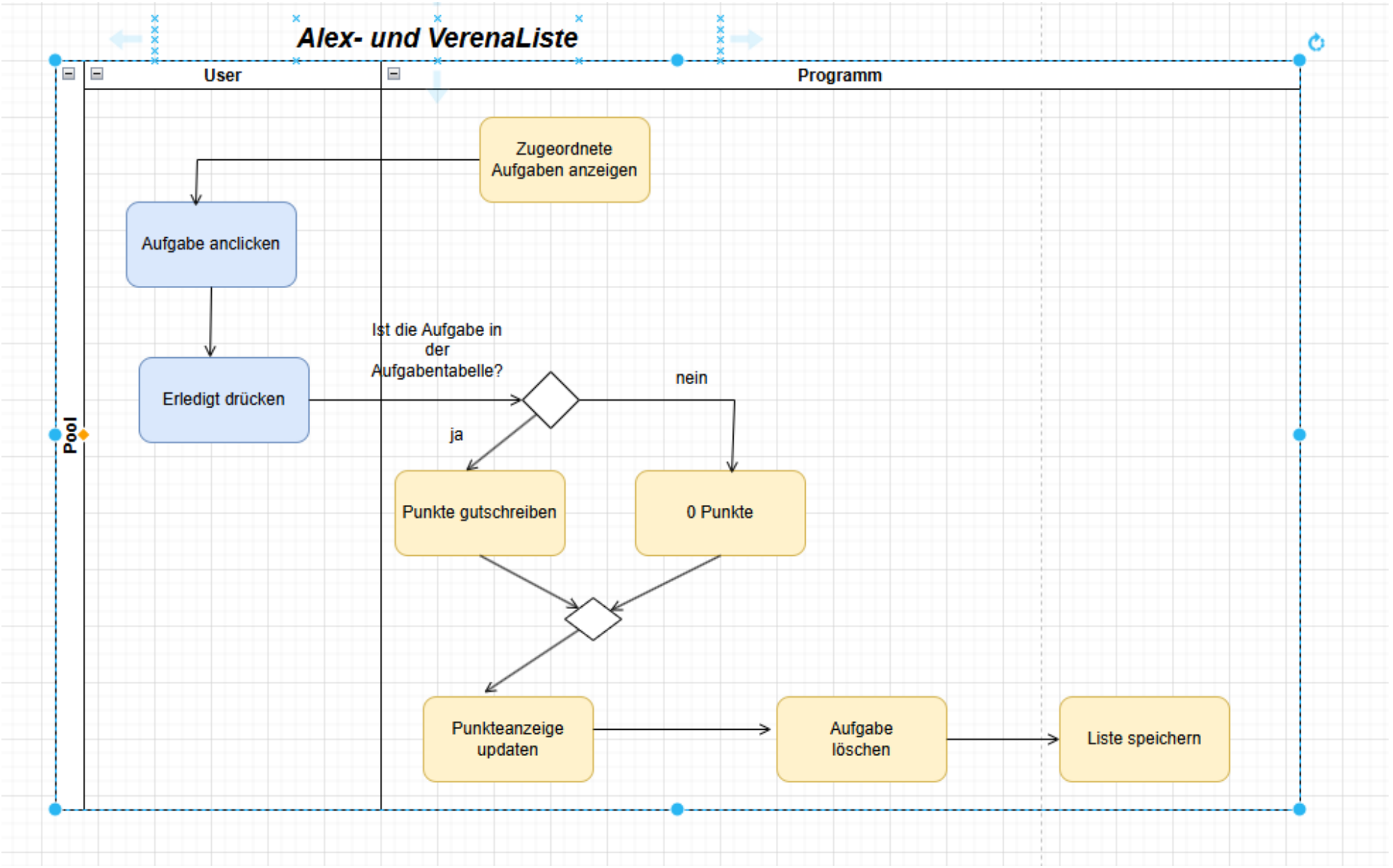


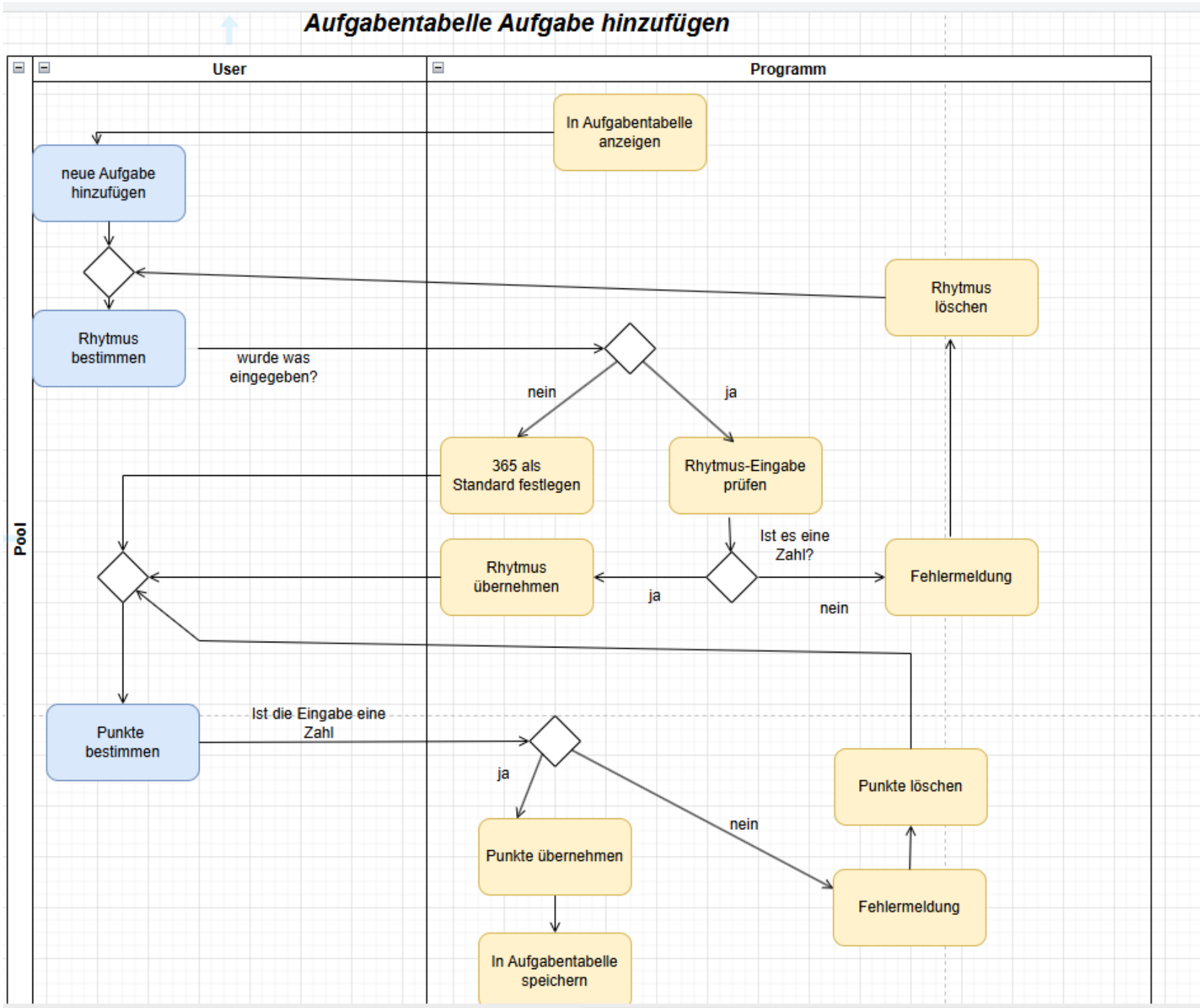
Programmstart

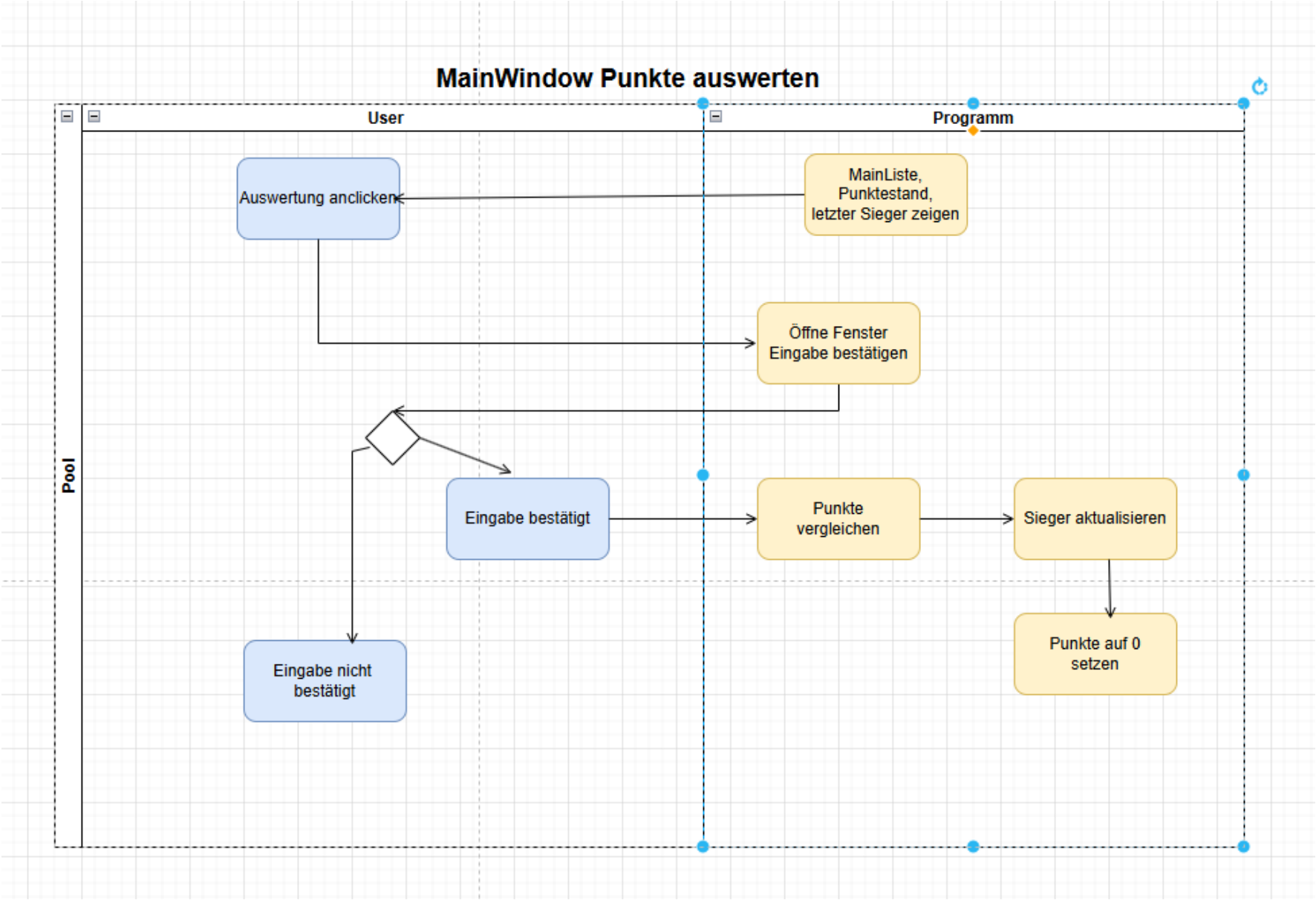



MainWindow Aufgaben hinzufügen
Pool
User
Programm
MainListe, Punktestand, letzter Sieger zeigen
Aufgabe beginnen einzutippen
Vergleiche Eingabe mit Aufgabentabelle ab
Eingabe bekannt?
nein
ja
Aufgabe fertig eingetippt
Vorschlag übernehmen
Mache logische Vorschläge
Aufgabe hinzufügen
Neue Aufgabe auf Mainliste darstellen
Mainliste speichern


MainWindow Aufgaben verschieben
Pool
User
Programm
Aufgabe anclicken
MainListe, Punktestand, letzter Sieger zeigen
Aufgabe auf Alex-Liste verschieben
Aufgabe auf Verena-Liste verschieben
Öffne Fenster Eingabe bestätigen
Eingabe nicht bestätigt
Eingabe bestätigt
Verschiebe auf Liste
Listen speichern

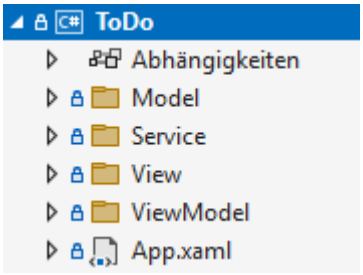

Alex- und VerenaListe
User
Programm
Pool
Zugeordnete Aufgaben anzeigen
Aufgabe anclicken
Erledigt drücken
Ist die Aufgabe in der Aufgabentabelle?
nein
ja
Punkte gutschreiben
0 Punkte
Punkteanzeige updaten
Aufgabe löschen
Liste speichern


Aufgabentabelle Aufgabe hinzufügen
Pool
User
Programm
In Aufgabentabelle anzeigen
neue Aufgabe hinzufügen
Rhytmus löschen
Rhytmus bestimmen
wurde was eingegeben?
nein
ja
365 als Standard festlegen
Rhytmus-Eingabe prüfen
Ist es eine Zahl?
Rhytmus übernehmen
Fehlermeldung
ja
nein
Punkte bestimmen
Ist die Eingabe eine Zahl
ja
Punkte löschen
nein
Punkte übernehmen
Fehlermeldung
In Aufgabentabelle speichern


MainWindow Punkte auswerten
Pool
User
Programm
Auswertung anclicken
MainListe,
Punktestand,
letzter Sieger zeigen
Öffne Fenster
Eingabe bestätigen
Eingabe bestätigt
Punkte
vergleichen
Sieger aktualisieren
Punkte auf 0
setzen
Eingabe nicht
bestätigt

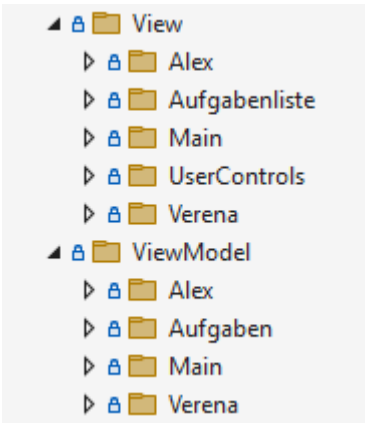
Projekt-Ordner-Struktur

Das Projekt wurde nach dem Model-View-ViewModel (MVVM) – Architekturprinzip aufgebaut, um eine saubere Trennung zwischen Benutzeroberfläche, Anwendungslogik und Datenmodel zu gewährleisten. Zusätzlich wurde ein Service-Ordner ergänzt, um wiederverwendbare Hilfsfunktionen auszulagern.



1. **View und ViewModel**

Sowohl der View- als auch der ViewModel-Ordner enthalten jeweils eigene Unterordner für die vier Hauptfenster der Anwendung:



- Mainliste
- Aufgabenliste
- Alexliste
- Verenaliste

Die Views beinhalten die grafische Oberfläche (XAML-Dateien), während die ViewModels die zugehörige Logik kapseln.

Im Unterordner UserControls befindet sich eine Vorlage für eine löschbare Textbox zur Wiederverwendung.

2. **Model**

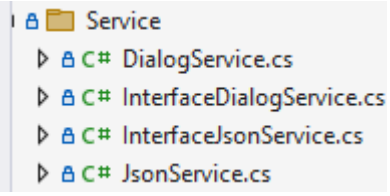
Im Model-Ordner sind die zentralen Datenklassen definiert:

- Die Klasse Aufgabe enthält Informationen über die Art der Aufgabe, den Tages-Rhythmus, die Punkteanzahl sowie das Datum, wann die Aufgabe zuletzt hinzugefügt wurde
- Die Kasse Eintrag verwaltet die konkreten Eintragungen auf den individuellen Listen der Personen

3. Services

Der Service-Ordner enthält Hilfsklassen für technische Funktionen:

- Der JSON-Service übernimmt das Speichern und Laden der Aufgabentabelle und des Listenstands
- Der DialogService kapselt die Anzeige von MessageBox-Dialogen
- Die zugehörigen Interfaces ermöglichen eine flexible Nutzung



Die Trennung der Speicherung der Listen, Punkte, Sieger und die Speicherung der Aufgabentabelle wurde bewusst gewählt, um eine spätere Umstellung auf SQL-Datenbanken zu vereinfachen.

4. App.xaml

Die Datei App.xaml wurde angepasst, um bestimmte Abläufe beim Starten und Beenden der Anwendung zu steuern – z.B. das automatische Speichern beim Beenden der Datei.

Verwendete Klassen und Methoden

Da die Erklärung und Auflistung aller Klassen und Methoden den Rahmen dieser Dokumentation sprengen würde, beschränke ich mich auf die wichtigsten Kern-Features.

Das Vorschlagssystem (MainViewModel)

Um die Eingabe der Aufgaben zu erleichtern, wurde ein dynamisches Vorschlagssystem implementiert. Es reagiert auf Benutzereingaben und schlägt über ein Dropdown-Menu automatische passende Aufgaben vor. Dies geschieht durch einen Abgleich der Eingabe mit der Aufgabentabelle.

Es besteht aus den folgenden Bestandteilen

1. EingabeText

Sobald in der Combobox eine Texteingabe erfolgt wird automatisch die Methode AktualisiereVorschläge() aufgerufen.

2. AktualisiereVorschläge()

Diese durchsucht die aktuelle Aufgabenliste (Verknüpfung über _aufgabenViewModel.Aufgabenliste) nach ähnlichen Begriffen und befüllt die Liste Vorschläge mit passenden Einträgen. Es wird nicht zwischen Groß- und Kleinschreibung unterschieden, auch die Anordnung - z.B. „Bad putzen“ oder „Reinigen des Bads“ wird beides bei der Eingabe „Bad“ gefunden - hat keine Auswirkung auf das Suchergebnis. Um die Möglichkeit der mehrfachen Treffer auszuschließen wurde Distinct() verwendet.

3. Vorschläge

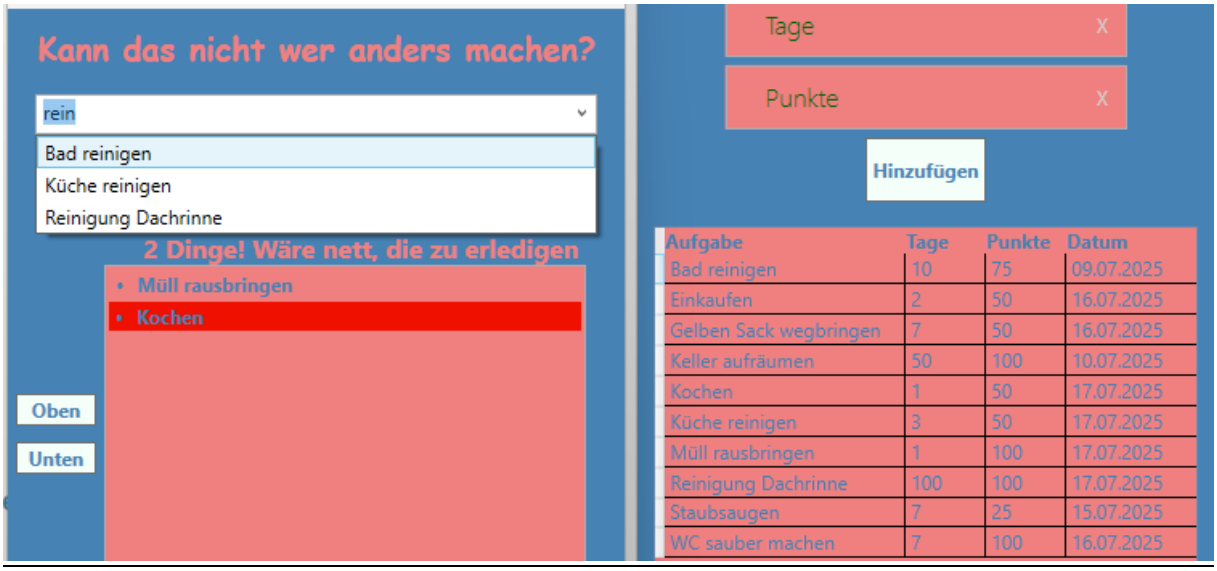
Eine ObservableCollection<string>, welche über ein Drop-down-Menu einfach abgefragt werden kann.

4. AusgewählterVorschlag

Wird ein Vorschlag aus der Liste ausgewählt, wird dieser automatisch als EingabeText übernommen, sodass der Nutzer dies nicht manuell übertragen muss.

Beispiel:

Durch die Eingabe „rein“ wird die Aufgabentabelle danach durchsucht und die Treffer „Bad reinigen“, „Küche reinigen“ und „Reinigung der Dachrinne angeben.



Per einfachen Click, kann der gesuchte Treffer übernommen werden.

Das „intelligente“ Hinzufügen (AufgabenViewModel)

Die Methode CheckHauptliste() aus dem AufgabenViewModel prüft, ob Aufgaben aus der allgemeinen Aufgabentabelle auf die Hauptliste gesetzt werden müssen – und stellt dabei sicher, dass keine Dopplungen entstehen.

1. Kontrolle der Fälligkeit

Jede Aufgabe ist in der Aufgabentabelle mit einem festen Rhythmus (Tage) und einem Datum, wann sie zuletzt hinzugefügt wurde (AufgabeHinzugefügt) hinterlegt. Sobald das Intervall seit dem letzten Hinzufügen überschritten wurde, gilt die Aufgabe als „fällig“.

2. Prüfung

Die Methode prüft, ob diese Aufgabe bereits in einer der drei Listen (Hauptliste, Alexliste, Verenaliste) vorhanden ist. Sollte das Ergebnis positiv sein, wird die Aufgabe nicht erneut hinzugefügt.

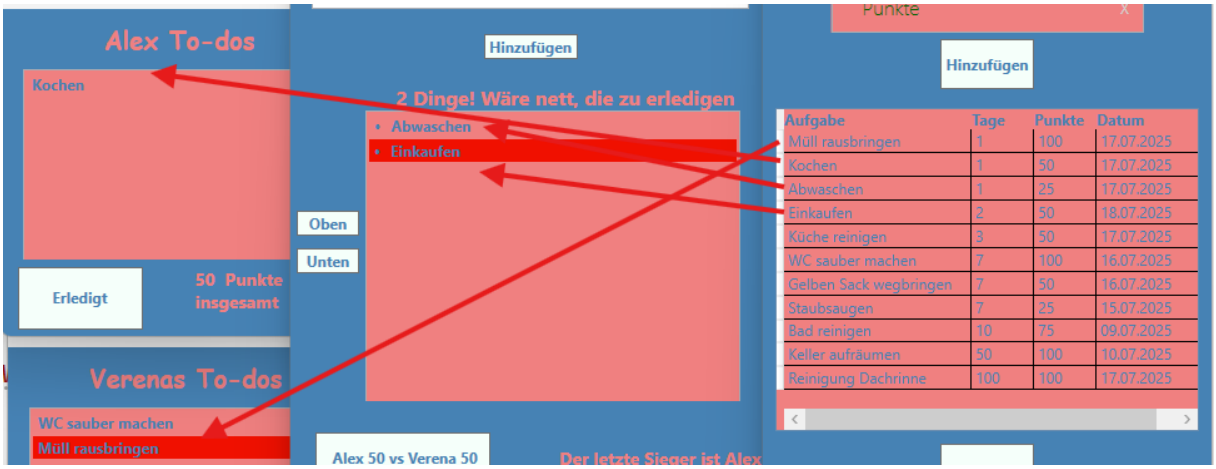
3. Hinzufügen zur Hauptliste

Falls eine Aufgabe fällig und in keiner Liste vorkommt, werden sie als neue Eintrag-Objekte zur Hauptliste hinzugefügt. Gleichzeitig erfolgt eine Anpassung des Datums bei AufgabeHinzugefügt auf den heutigen Tag, um das Intervall neu zu starten.

4. Speichern

Zum Abschluss erfolgt ein Speichern(), um die neuen Aufgabenzustände zu sichern.

Beispiel:



Am heutigen 18.07.2025 hätten die Aufgaben:

- Müll rausbringen
- Kochen
- Abwaschen
- Einkaufen

Auf der Hauptliste hinzugefügt werden sollen, wie man am jeweiligen Datum- und Tage-Eintrag ablesen kann. Da jedoch „Abwaschen“ bereits auf der Hauptliste, „Kochen“ auf der Alexliste und „Müll rausbringen“ auf der Verenaliste waren, wurde lediglich „Einkaufen hinzugefügt.

DialogService

Im ViewModel wurden zunächst MessageBoxen vielfach eingesetzt, was zu Problemen bei den Unit-Tests führte. Die Lösung war eine zentrale Regelung der Anzeige der Nachrichten und Rückfragen über das DialogService. Es Implementiert das Interface aus InterfaceDialogService und kapselt die Anzeige von MessageBox-Dialogen.

„ShowMessage(string message, string title = „!!!““ zeigt eine einfache Informationsmeldung mit OK-Button.

„ShowYesNo(string message, string title = „???““ zeigt eine Ja-Nein-Frage und gibt ein „true“ zurück bei der Auswahl „Ja“.

Die Methoden können über das InterfaceDialogService eingesetzt werden und verhindern eine Abhängigkeit von der MessageBox.

Neben den bereits erwähnten Vorteil der Testbarkeit, wird durch den DialogService auch eine Trennung von Logik und UI erreicht.

JsonService

Die Klasse JsonService ist für das Speichern und Laden der Anwendungsdaten zuständig. Sie nutzt das JSON – Format zur strukturierten Ablage der Aufgabentabelle und der Benutzerlisten. Es hat die Aufgaben:

1. Speichern und Laden der Listen, Punktestand und Gewinner
2. Separate Speicherung der Aufgabentabelle
3. Fehlerbehandlung
4. Anpassung des Datumsformats

Die Daten werden in den zwei Dateien daten2.json und aufgaben.json gespeichert.

Die Datumskonvertierung muss sowohl beim Speichern als auch beim Laden gemacht werden, da die Datumsangabe im US-Format hinterlegt und angezeigt wird.

WriteIndented = true ermöglicht eine manuelle Lesbar- und Editierbarkeit.

Sollte die Datei beschädigt oder nicht lesbar sein, werden automatisch leere Listen erstellt und eine Fehlermeldung an den Benutzer herausgegeben.

Tests

Für die Anwendung wurden gezielt vier Funktionalitäten mittels Unit-Tests überprüft:

- Das korrekte Aktualisieren der Listenüberschrift
- Das manuelle Hinzufügen auf der Hauptliste
- Das Verschieben der einzelnen Listenelemente
- Das korrekte Hinzufügen der Aufgaben auf die Hauptliste

Testvorbereitung:

Es wurde ein Verweis auf das Hauptprojekt eingefügt und die erforderlichen NuGet-Pakete installiert:

Microsoft.Net.Test.Sdk, NUnit, NUnit.Analyzers und NUnit3TestAdapter.

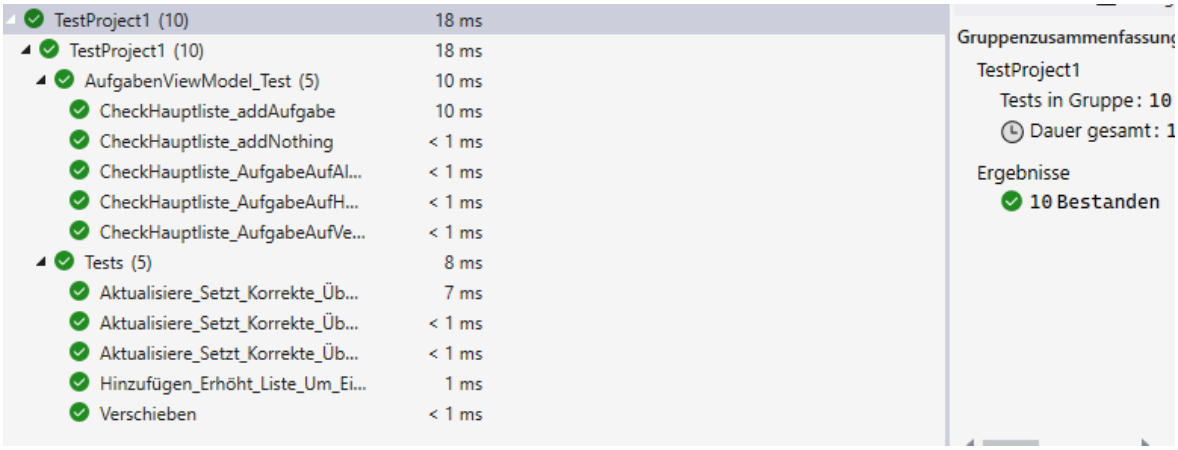
Da die Anwendung auf JSON-Dateien und Benutzerabfragen über MessageBoxen basiert, musste für die Testumgebung passende Hilfsdateien erstellt werden.

- In der Datei TestDialog.cs wurde festgelegt, dass alle Ja/Nein-Abfragen immer mit „Ja“ also „True“ beantwortet werden.
- Für die Tests wurde des Weiteren im TestJsonService.cs festgelegt, dass man mit leeren Listen startet und Aufgaben lediglich Art und Datum als Spalteneinträge kennen.

Durchführung und Ergebnisse:

Alle Tests wurden erfolgreich abgeschlossen. In UnitTest1.cs wurden die verschiedenen Fälle der Listenüberschrift getestet (leer, wenig, viele Einträge). Außerdem wurde überprüft, ob Elemente korrekt hinzugefügt verschoben werden. Es wurden auch Grenzfälle getestet, wenn man z.B. das oberste Element weiter nach oben verschieben möchte.

In AufgabenViewModel-Test.cs lag der Fokus auf dem Verhalten beim Abgleich mit den Benutzerlisten. Es wurde geprüft, ob Aufgaben korrekt erkannt und nur dann zur Hauptliste hinzugefügt werden, wenn sie noch nicht zugeordnet sind.



Erweiterbarkeit

Das Programm ist modular aufgebaut und bietet dadurch vielfältige Möglichkeiten zur Erweiterung. Einige naheliegende Verbesserungen und zukünftige Entwicklungsschritte sind nachfolgend aufgeführt:

Umstellung der Datenspeicherung:

Die derzeitige Speicherung erfolgt im JSON-Format. Eine Umstellung auf eine **relationale Datenbank** z.B. SQLite wird dringend empfohlen, da es wiederholt zu Fehlzuweisungen in den Listen (Alex / Verena) kam. Eine Datenbanklösung würde hier mehr Stabilität und Datenintegrität bieten.

Erweiterte Statistikfunktionen:

Zur Motivation und besseren Nachvollziehbarkeit könnten die gesammelten Punkte detailliert aufgeschlüsselt werden. Denkbar wären auch:

- Eine Historie, welche Person wann welche Aufgabe erledigt hat
- Eine Häufigkeitsauswertung, wie oft bestimmte Tätigkeiten im Haushalt übernommen wurden
- Monatliche Punktediagramme

Benutzerverwaltung & Login

Eine optionale Login-Funktion wäre sinnvoll, wenn die jeweiligen Listen vor gegenseitigem Zugriff geschützt werden soll. Dies kann auch die die Grundlage für eine späteres **Multi-User-System** bieten.

Aufgabentabelle

Momentan speichert das Programm lediglich, wann eine Aufgabe zur Hauptliste hinzugefügt wurde. Zukünftig sollte zusätzlich das **Erledigungsdatum** einer Aufgabe festgehalten werden. Dieses Datum könnte dann als Startpunkt für das Wiederholungsintervall dienen und dadurch die automatische Planung weiter verbessern.

Undo-Button

Auch wenn die wichtigsten Tätigkeiten vom User in der **MessageBox** bestätigt werden müssen, ist ein Undo-Button also ein **rückgängig** machen, eine sinnvolle Ergänzung.