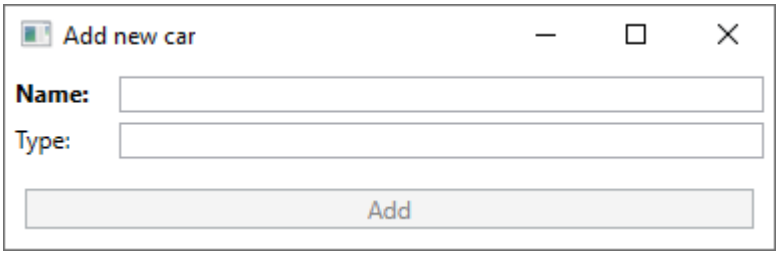
C# Telemetry API Client – Vorgehensweise

- Anlegen eines Views, ViewModel-Klasse und Kopieren des Car-Models aus der Web-Applikation
    - Im Car-Model die EF-Annotations entfernt, TelemtryData-Liste entfernt
    - Im Car-Model formatierte Properties angelegt: „CreatedAtFormatted“ und „ModifiedAtFormatted” (mit CurrentCulture, damit richtiges Datumsformat verwendet wird)
    - [JsonIgnore]-Annotations für Formatted-Properties
    - App.xaml angepasst (Start-XAML-Datei gesetzt)
- NuGet-Pakete: Newtonsoft.Json, RestSharp, Fody, PropertyChanged.Fody (diese zwei sind für automatische Property-Changed-Properties) und WpfAnimatedGif (Um Ladeanimation als GIF-Datei abzuspielen)
- Konstanten-Klasse angelegt
- Ladeanimation implementiert
    - **Anmerkung:** Um die Animation gut zu sehen, kann in den Konstanten die URL zur API geändert werden, sodass versucht wird eine Verbindung herzustellen bis zum Timeout-Limit.
- Asynchrone Methoden für API-Zugriff erstellt (im ViewModel)
    - LoadAllCars(): Lädt alle Cars
    - DeleteCar(Car toDelete): Löscht ein Car-Model
- ListView: ItemTemplate aufgebaut
- ListView: Kontextmenü erstellt
    - Commands im ViewModel angelegt. Für das Löschen soll eine Prompt angezeigt werden; dazu wird ‚System.Windows.MessageBox‘ verwendet. Weil die Prompt alt aussieht, kann ein Manifest erstellt werden (https://stackoverflow.com/questions/27773492/wpf-messagebox-looks-unstyled-while-windowsforms-messagebox-looks-good). Rechtsklick auf Projekt → Neues Element → Anwendungsmanifestdatei. In dieser muss der „dependecy“-Teil auskommentiert werden (siehe Link).
- Dialog für das Hinzufügen eines Autos erstellt: View und ViewModel. Command für „Add“ sieht folgendermaßen aus:

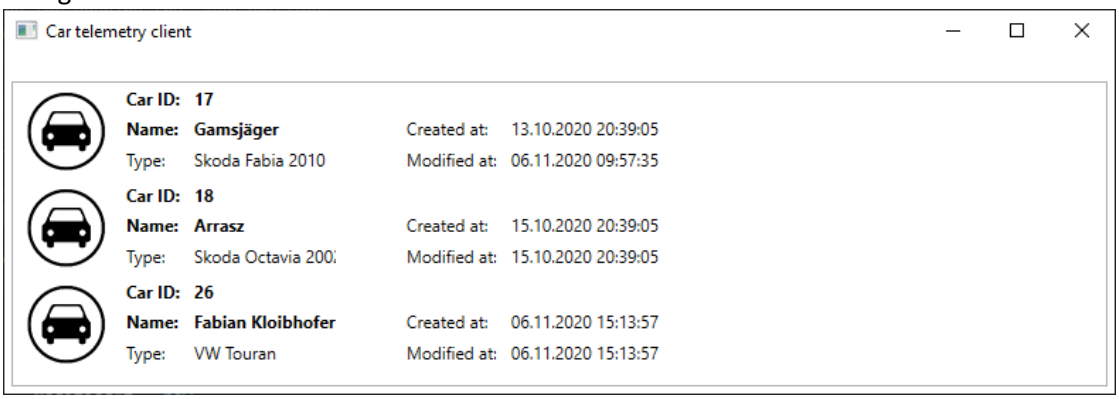
```
/// <summary>
/// Command for opening the 'AddCarDialog'.
/// </summary>
public ICommand AddCarCommand => new RelayCommand((e) =>
{
    AddCarViewModel addCarViewModel = new AddCarViewModel();
    AddCarDialog addDialog = new AddCarDialog(addCarViewModel);
    addDialog.ShowDialog(); // Blocks main window

    // Exit by ok == valid input and user clicked add
    if (addCarViewModel.ExitByOk)
    {
        DateTime now = DateTime.Now;
        Car newCar = new Car()
        {
            Name = addCarViewModel.Name,
            Type = addCarViewModel.Type,
            CreatedAt = now,
            ModifiedAt = now
        };

        Task.Run(() => AddCar(newCar));
    }
},
c => !ErrorOccured);
```

    - Zuerst wird ViewModel angelegt, dann die View angezeigt.

  - Wenn „ExitByOk“ true ist, signalisiert das erfolgreiche Beenden des Dialogs (korrekte Eingaben und Add geklickt).
  - Car anlegen und an API senden (per Methode).
- Dialog kann nicht bestätigt werden, wenn nicht beide Felder ausgefüllt werden:



- Fertiger Client:



- Ladeanimation:

- Errorfeld: