Wetr-Plattform

Studienprojekt Software-Engineering zu SWK5

Johannes Naderer

Wintersemester 2018/2019

Inhalt

Disclaimer

Dieses Dokument, inklusive aller Texte und Abbildungen, sowie der mitgelieferte Programmcode sind lediglich Work-In-Progress Implementierungen und unterstehen häufigen Änderungen.

Die aktuellen Dokumente sind auf Github unter folgendem Link erreichbar.

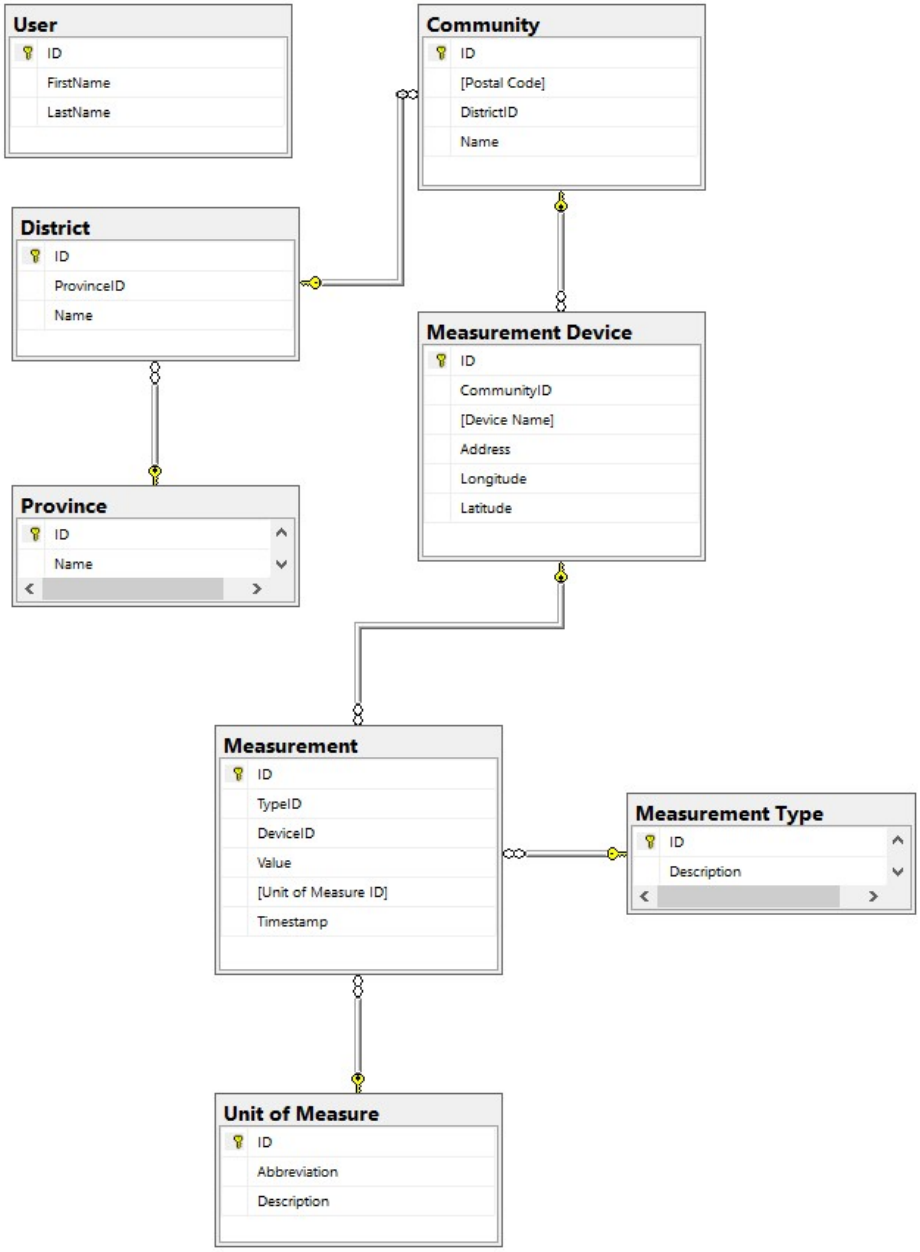
https://github.com/swk5-2018ws/wetr-bb-naderer-bb

Wetr.Server

Datenmodell

Das verwendete Datenmodell hält sich streng an die nötigen Entitäten. Hier wird angenommen, dass Benutzer lediglich zur Verwaltung und Auswertung der Messdaten berechtigt sind, und nicht direkt mit den Wetterstationen bzw. deren Messungen in Verbindung stehen.

Datenmodell – Übersicht:



Data-Access-Layer

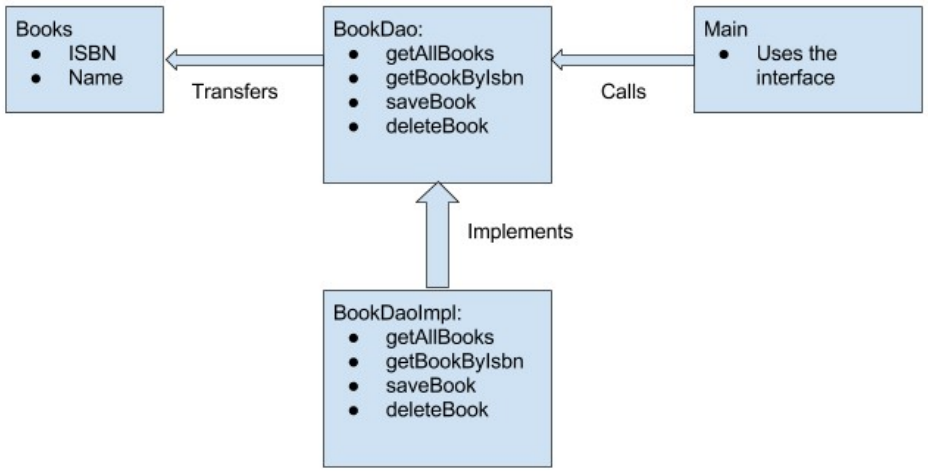
Die Datenzugriffsschicht wird mithilfe des DAO-Patterns abgebildet (siehe http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/dao.html). Dadurch wird der letztendliche Datenzugriff auf die verwendete Microsoft SQL Server Datenbank durch vorgestellte Interfaces abstrahiert.

Wetr.Server.DAL.IDAO enthält diese Interface-Klassen, die jeweils Methoden definiert, über die die verwendeten Domänenklassen empfangen und eingefügt werden können. Der Nutzer der Datenzugriffslogik verwendet lediglich diese Interfaces für den Zugriff.

Wetr.Server.DAL.DAO enthält die entsprechenden Implementierungen zu den in IDAO definierten Interfaces. Für den Datenbankzugriff wird ADO.NET verwendet. Für die Erstellung der Datenbankverbindung und das Ausführen der letztendlichen SQL-Statements werden die Hilfsklassen aus Wetr.Server.Common verwendet. Durch die Abstraktion der Implementierung, lassen sich DAO-Implementierungen einfach austauschen.

Wetr.Server.DAL.DTO enthält die Domänenklassen, die den entsprechenden Entitäten der Datenbank entsprechen.

Skizzierung DAO Pattern anhand eines Beispiels, Quelle https://www.journaldev.com/16813/dao-design-pattern



Books steht in der Grafik stellvertretend für die Domänenklassen in Wetr.Server.DAL.DTO.
BookDao steht für die Interfaces in Wetr.Server.DAL.IDAO
BookDaoImpl steht für die Implementierungen der Interfaces in Wetr.Server.DAL.DAO

Die Interfaces sind entsprechend kurzgehalten, um zukünftig einfach Erweiterbar zu sein.

Die generierten Testdaten für das Projekt sind im Git-Repository unter https://github.com/swk5-2018ws/wetr-bb-naderer-bb/tree/master/Wetr.Server/Db zu finden. Hier finden sich sowohl reine Testdaten als auch das SQL-Schema dafür.

Um zukünftigen Clients dieser Serverapplikation entsprechend nötige Funktionalität zur Verfügung zu stellen, wurden mit den Projekten Wetr.Server.BL.IDefinition bzw. Wetr.Server.BL.Implementation ent entsprechende Schnittstellen definiert, damit Clienten diese nutzen können, ohne statisch von konkreten Implementierungen abzuhängen.

Tests

Als Testframework wird xUnit eingesetzt https://xunit.github.io/.
Um xUnit für Full-Framework Projekte einzusetzen sind einige Vorschritte nötig, um die entsprechenden Nuget-Pakete für die Verwendung zu erhalten. Der Vorgang mit VisualStudio ist hier dokumentiert: https://xunit.github.io/docs/getting-started-desktop

Für die Tests der Datenzugriffsschicht werden einfache Unit-Test-Methoden verwendet. Da sich die einzelnen Methoden (FindAll, GetById, Insert, …) am besten im Verbund testen lassen (Insert -> danach Prüfung via FindAll, …) wurde je DAO-Klasse eine Testmethode implementiert, die bereits mehrere Methoden prüft.

Die Tests sind im Projekt Wetr.Server.Tests zu finden.

Wetr.Clockpit

Übersicht

Wetr.Cockpit soll den Haupteinstiegspunkt für Administratoren der Wetr-Plattform darstellen. In Wetr.Cockpit können Wetterstationen verwaltet (CRUD) und ausgewertet werden.

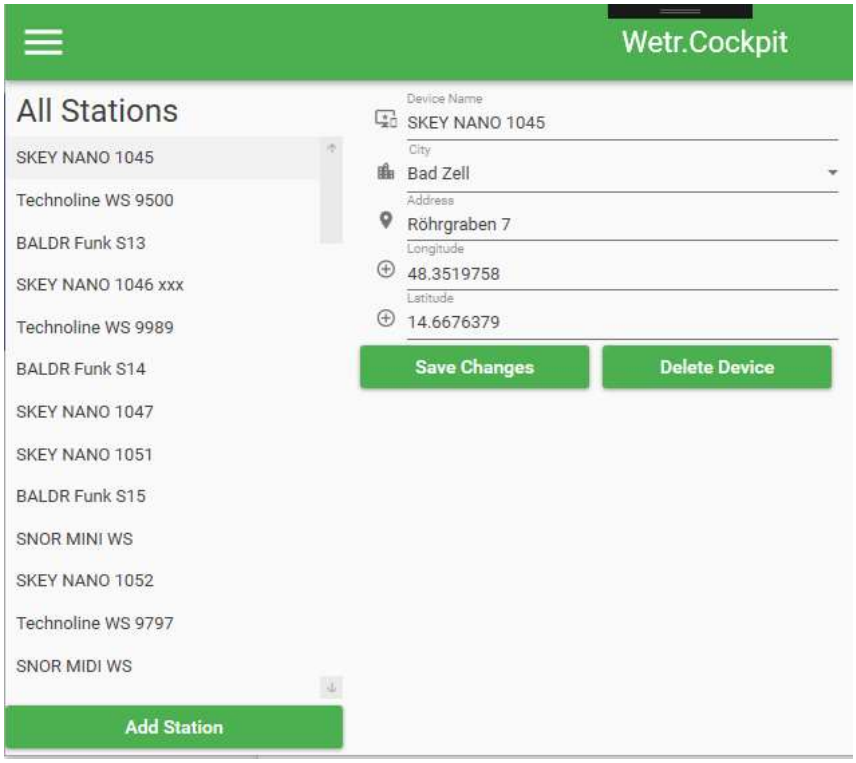
Der Windows-Client wurde mithilfe der WPF und unter Verwendung des MVVM Musters entwickelt.

Um dem Nutzer eine möglichst übersichtliche und moderne Oberfläche zu bieten, wird für Wetr.Cockpit die Fremdbibliothek MaterialDesignInXAML verwendet. Die Bibliothek hilft dabei, Benutzeroberflächen in Anlehnung an die von veröffentlichten Design-Richtlinien von „MaterialDesign“ zu erstellen. Infos bzw. Anleitungen dazu sind unter den folgenden Links einzusehen:

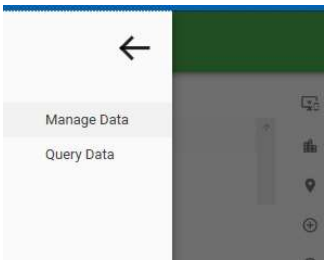
https://material.io/design/

http://materialdesigninxaml.net/

Beim Öffnen der Applikation wird der Nutzer direkt in die Verwaltungsmaske geführt. Hier sind zur Linken die Verfügbaren Wetterstationen zu finden, im Hauptteil befindet sich die Detailansicht, aus der Informationen entnommen und abgeändert werden können. Hier können Stationen auch gelöscht werden, sofern für die Wetterstation keine Messdaten vorliegen.



Um zur Hauptfunktionalität von Wetr.Cockpit zu gelangen, muss über das Burger-Menü navigiert werden



.

Nach der Betätigung von „Query Data“ wird direkt in die Abfragemaske gewechselt.

Über die Verfügbaren Filtermöglichkeiten in der oberen Hälfte der Maske lassen sich schnell Filterkriterien festlegen, anhand denen die Datenbank abgefragt wird.

Die Ergebnisse werden nach Vorliegen der Ergebnisse entsprechend im unteren Teil angezeigt. Neben der einfachen Abfrage von einzelnen Messungen zu einer bestimmten Station (Abbildung 1) lassen sich so auch Gruppenweise aggregierte Abfragen erstellen.

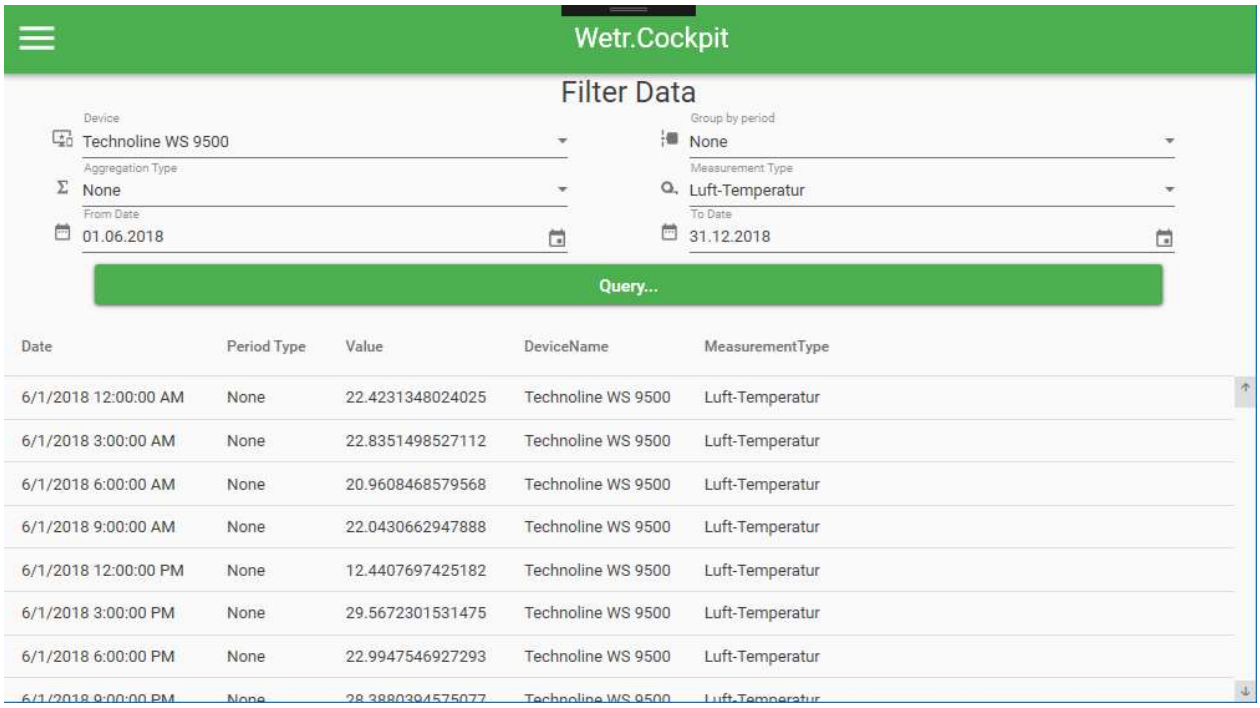


Abbildung 1

Sämtliche Eingaben werden direkt bei der Eingabe validiert, sollten nicht valide Zustände auftreten, wird der Nutzer sofort darüber informiert (Abbildung 2).

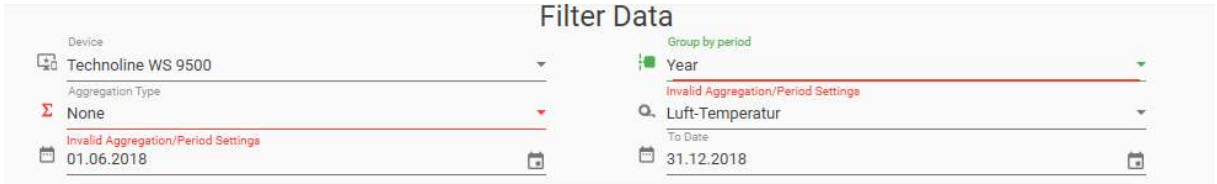


Abbildung 2

Implementierungsdetails

Das Hauptfenster selbst besteht im Grunde genommen lediglich aus dem Burger-Menü und einem ContentControl, in dem die Oberfläche zur jeweils ausgewählten Option angezeigt wird.

Hier ist insteressant, dass das ViewModel dahinter („*MainListItem*“) nur aus dem Namen des Menüeintrags und einem UserControl besteht. Somit hat der Menüpunkt selbst die dazugehörige Oberfläche auf Objektbasis in sich selbst integriert. Die Anzeige erfolgt dann über Datenbindung des UserControls auf das ContentControl des Hauptfensters.

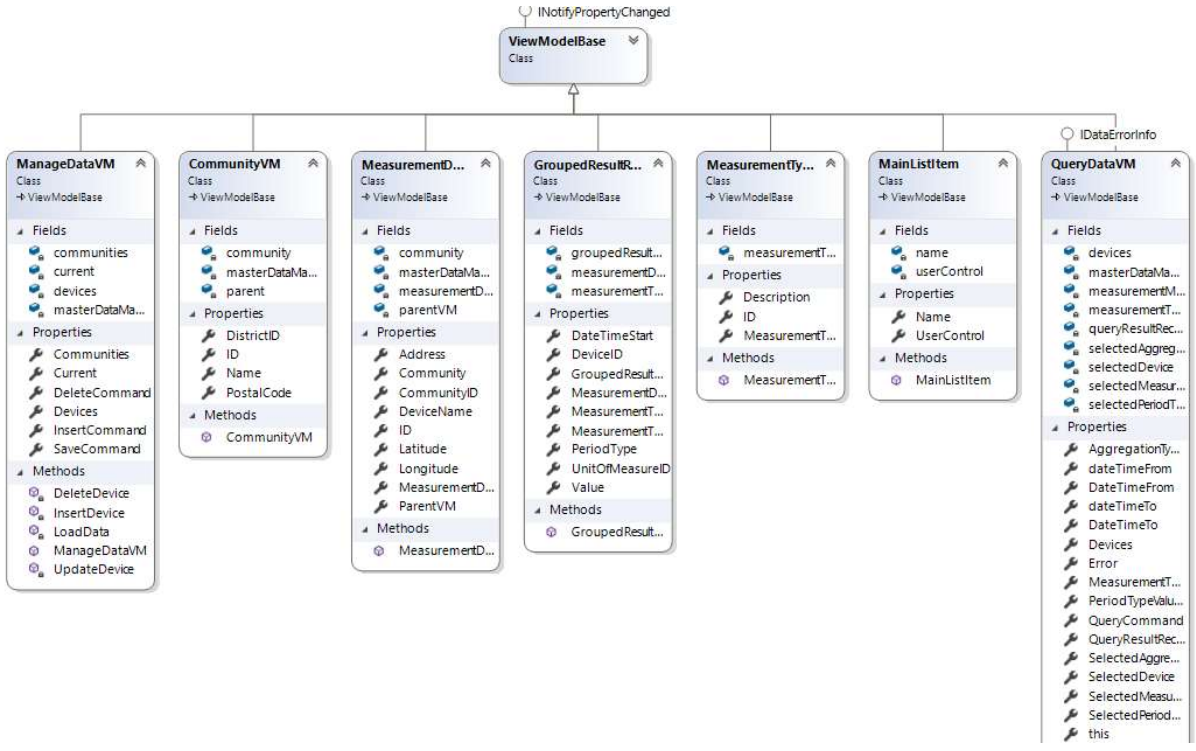
Die Validierung der Controls in den einzelnen Oberflächen wurde großteils durch Nutzung der Schnittstelle *IDataErrorInfo* erstellt, welches das Überschreiben eines Indexers fordert. Infos zu *IDataErrorInfo* sind unter folgendem Link zu finden:

https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.idataerrorinfo?view=netframework-4.7.2

Der Datenzugriff wird ausschließlich über die in Ausbaustufe 1 definierten DAO’s und somit von Datenbank bishin zur GUI komplett asynchron abgewickelt.

Für die Implementierung wurden weiters die aus dem Vorlesungsbetrieb gelehrten Konzepte von RelayCommand und INotifyPropertyChanged verwendet.

Um alle Daten aus der Geschäftslogik entsprechend in MVVM einzubetten, wurden mehrere ViewModel Klassen erstellt. Schematische Übersicht:



Wetr.Simulator

Übersicht

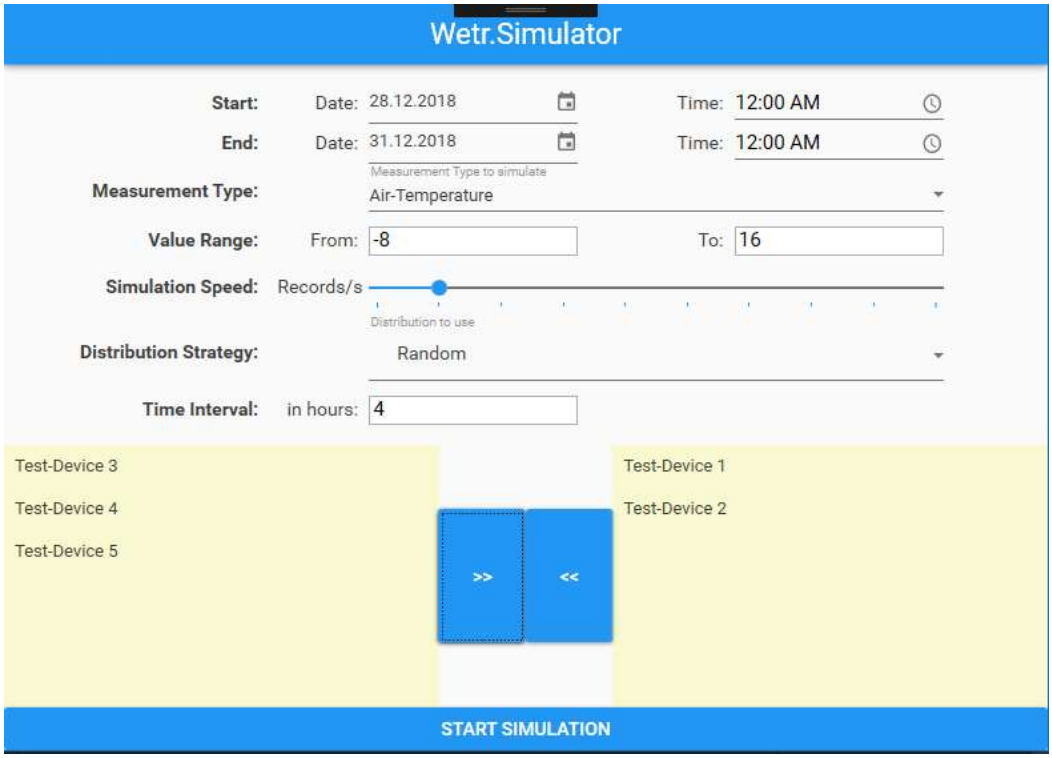
Wetr.Simulatur bietet den Nutzern der Wetr-Plattform einen einfachen und benutzerfreundlichen Weg, verschiedene Wetterdaten auf mehrere Arten zu generieren.

Wie auch Wetr.Cockpit wurde der Simulator mithilfe der WPF entwickelt und basiert ebenfalls auf dem MaterialDesign.

Wetr.Simulator ist eine eigenständige Applikation, die lediglich via Webservices mit der Datenbank spricht. In Ausbaustufe 2 ist diese Kommunikation aufgrund der fehlenden Services gefaked.

Um ein einheitliches Look&Feel zu garantieren, wird wie auch im Cockpit MaterialDesignInXAML eingesetzt. Um das Reiter-Layout für die Simulationsansicht zu erstellen, wurden hier jedoch noch die Fremdbibliotheken *Dragablz* und *MahApps* mit eingebunden, die MaterialDesignInXAML um Reiteroberflächen und verschiebbare Controls erweitert.

Beim Öffnen der Applikation wird der Nutzer direkt zur Simulationskonfiguration geführt. In diesem Fenster können Simulationsreihen parametrisiert und direkt ausgeführt werden.



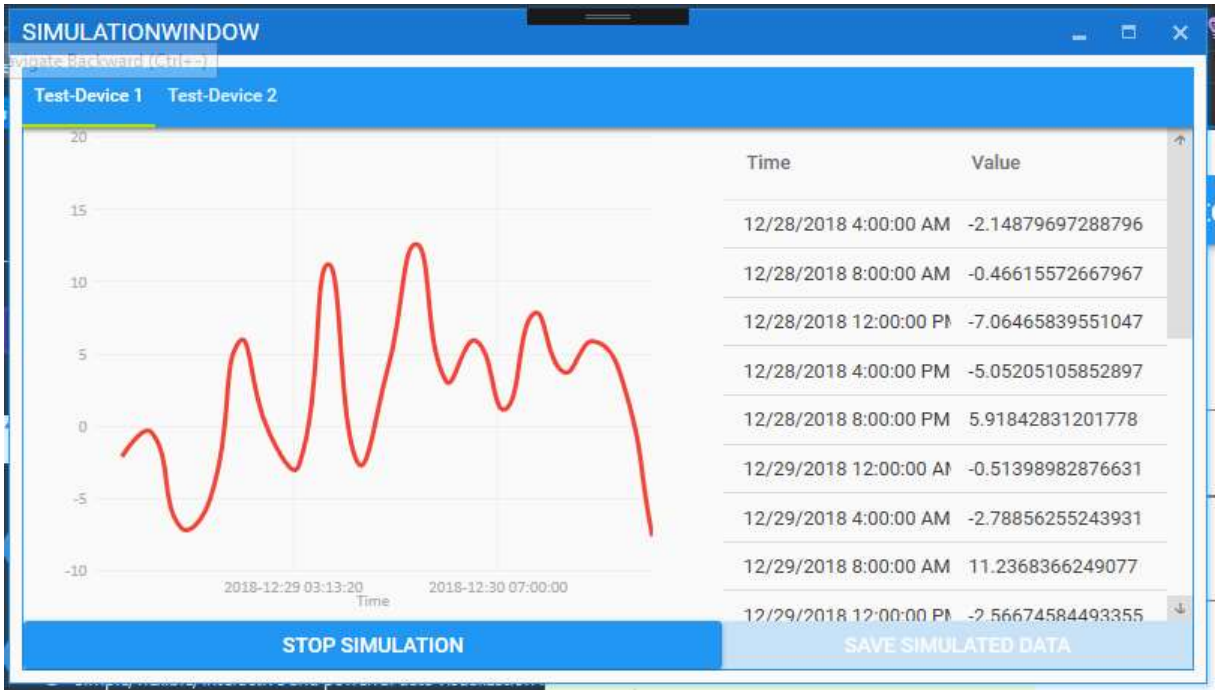
Im oberen Teil der Maske befinden sich sämtliche Konfigurationsmöglichkeiten für Simulationsreihen.

- Start & End: Hier wird festgelegt, für welchen Zeitraum Wetterdaten simuliert werden sollen.
- Measurement Type: Festlegung, welche Kenngröße simuliert werden soll.
- Value Range: Vergabe von Wertminimum und Maximum, die die Simulation respektiert.
- Simulation Speed: Kontrolliert die Simulationsgeschwindigkeit, hier kann diskret zwischen einem und bis zu 10 Datensätzen pro Sekunde gewählt werden.
- Distribution Strategy: Gibt die Verteilungsstrategie für die Werte an.
- Time Interval: Bestimmt in welchem (Echt) - Zeitabstand Daten generiert werden sollen.4 Stunden heißt hier z.B. dass für alle 4 Stunden ein Datensatz erstellt wird. Also im Beispiel lt. Abbildung um 12:00, um 16:00 usw.

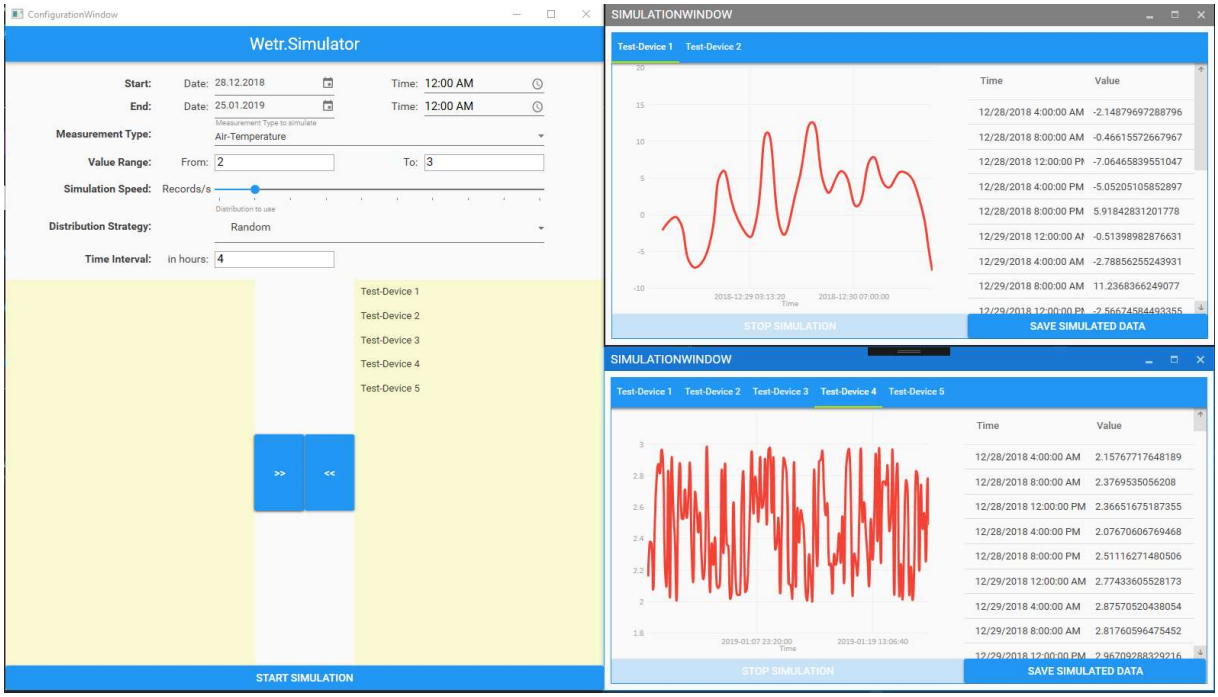
Die beiden Listen am unteren Ende der Oberfläche stellen die verfügbaren bzw. bereits selektierten Geräte für die Simulation dar.

In der linken Hälfte befinden sich alle verfügbaren Geräte. Diese können via Betätigung des „>>“ Knopf einfach zur Liste der zu simulierenden Geräte hinzugefügt werden.

Sobald „Start Simulation“ betätigt wird, wird für alle selektierten Geräte anhand der Einstellungen die Simulation gestartet.



Die Simulation selbst wird hier in einem eigenen Fenster gestartet. So ist es möglich mehrere Simulationen gleichzeitig laufen zu lassen, da das Konfigurationsfenster während laufenden Simulationen voll funktionsfähig bleibt, und weitere Simulationen starten kann.



Im Simulationsfenster wird für jedes ausgewählte Gerät eine eigene Zahlenfolge generiert, die dann direkt mithilfe der Listenansicht und des Diagramms verfolgt werden kann.

Zwischen den einzelnen Geräten kann einfach und schnell über die entsprechenden Reiter im oberen Teil der Simulation umgeschaltet werden. Diese Reiter lassen sich darüber hinaus via Drag&Drop einfach neu anordnen und verschieben.

Die Anzeige des Diagramms wurde mithilfe der Fremdbibliothek LiveCharts erstellt.

Links zu Fremdkomponenten:

http://materialdesigninxaml.net/

https://mahapps.com/

https://dragablz.net/

https://lvcharts.net/

Implementierung

Sämtliche Konfigurationen aus der Konfigurationsansicht werden im Hintergrund in einem einzigen ViewModel zusammengefasst „*ConfiguratorVM*“. Sobald der Benutzer den Start der Simulationen anstößt passieren mehrere Dinge:

1.) Aus der Gesamtheit der vorhandenen Informationen in *ConfiguratorVM* werden die für die Simulation relevanten Parameter in ein dafür vorgesehenes Objekt der Klasse „*SimulatorConfiguration*“ verpackt.
2.) Diese Konfiguration wird dann gemeinsam mit der Liste der ausgewählten zu simulierenden Geräte („*MeasurementDeviceVM*“) an ein neu erstelltes Simulationsfenster übergeben.
3.) Das Simulationsfenster erstellt nun für jedes übergebene Gerät eine Instanz von „*SimulatedMeasurementDeviceVM*“. Dieses erbt vom übergebenen Gerät, und erweitert es um die für die Simulation nötigen Erweiterungen
    - *ChartValues*: Behälter für generierte Daten für das Gerät
    - *Config*: Hier wird die Simulationskonfiguration für das Gerät gespeichert
    - *AxisMin*, *AxisMax*, *Formatter*: Informationen zum Zeichnen des Diagramms
    - *Timer*: Der *DispatcherTimer* zum Generieren der Simulationswerte
4.) Die Timer für alle zu simulierenden Geräte werden gestartet, diese nutzen die statische Klasse „*DataGenerator*“ um anhand der Übergabeparameter aus der Simulationskonfiguration neue Werte zu generieren.

Die generierten Daten werden via Data-Binding direkt an die entsprechenden GUI Komponenten gebunden und in Echtzeit dargestellt.

Für die Kommunikation mit dem Rest Service wurde ein Fake-Interface samt Implementierung in *IRestClient* bzw. *MockRestClient* erstellt.

Schematische Übersicht über die verwendeten ViewModels: