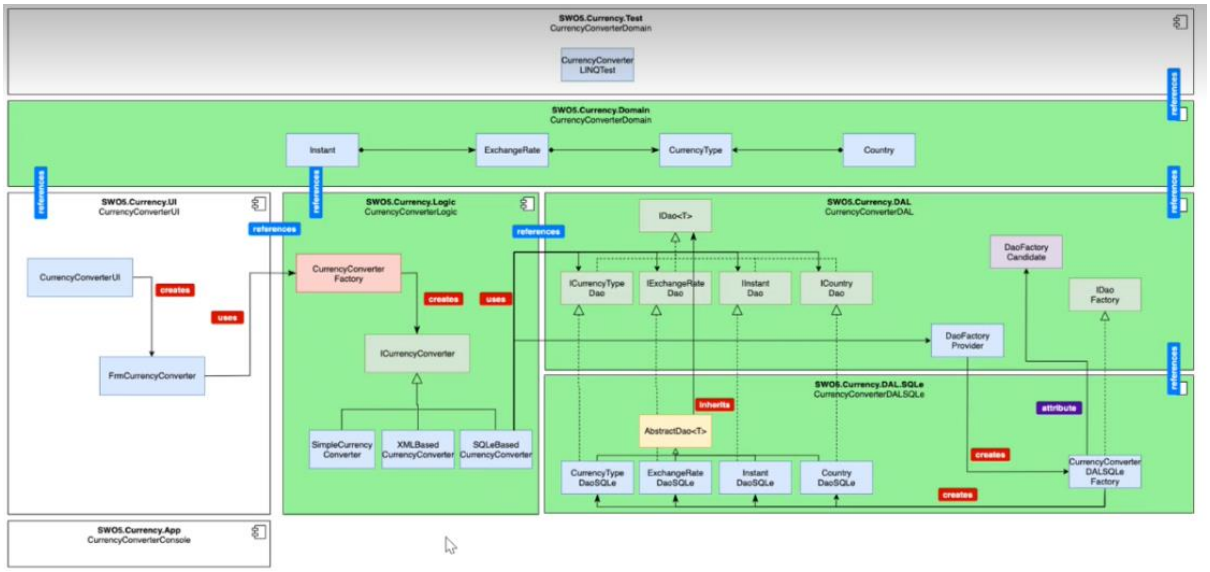
SWO5 - UE01

Aufwand: 20h

1. Architekturbeschreibung:

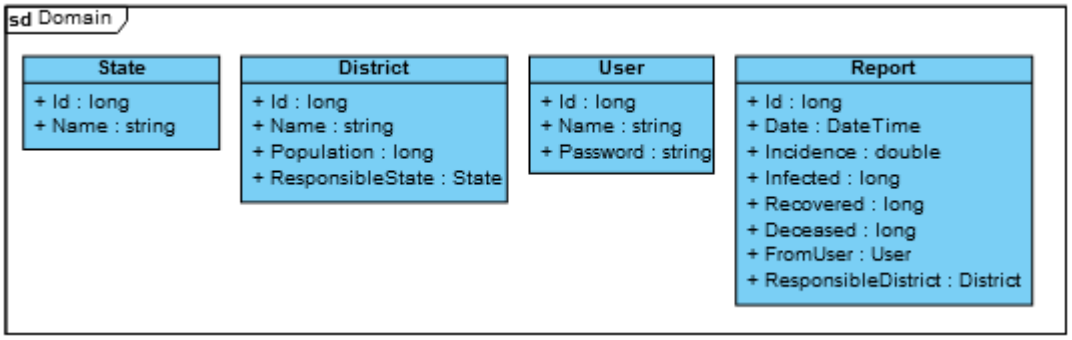
Das Projekt orientiert sich stark an demjenigen Projekt, welches im Laufe der Übung durchbesprochen wurde, weshalb der Großteil der Struktur gleich ist. Zum Vergleich möchte ich das in der Übung durchbesprochene Projekt nochmals zeigen:



Die zwei größten Änderungen stellen die Komponenten „Domain“ und „Logic“ dar, was auch Sinn macht, da das grundlegende Gerüst gleich ist, während sich aber die Daten und die Anwendungsfälle ändern. Ansonsten ändern sich nur die Namen für die Anwendungsfälle und Domänenobjekte und der Fakt, dass in diesem Fall noch keine GUI/App zu entwickeln war.

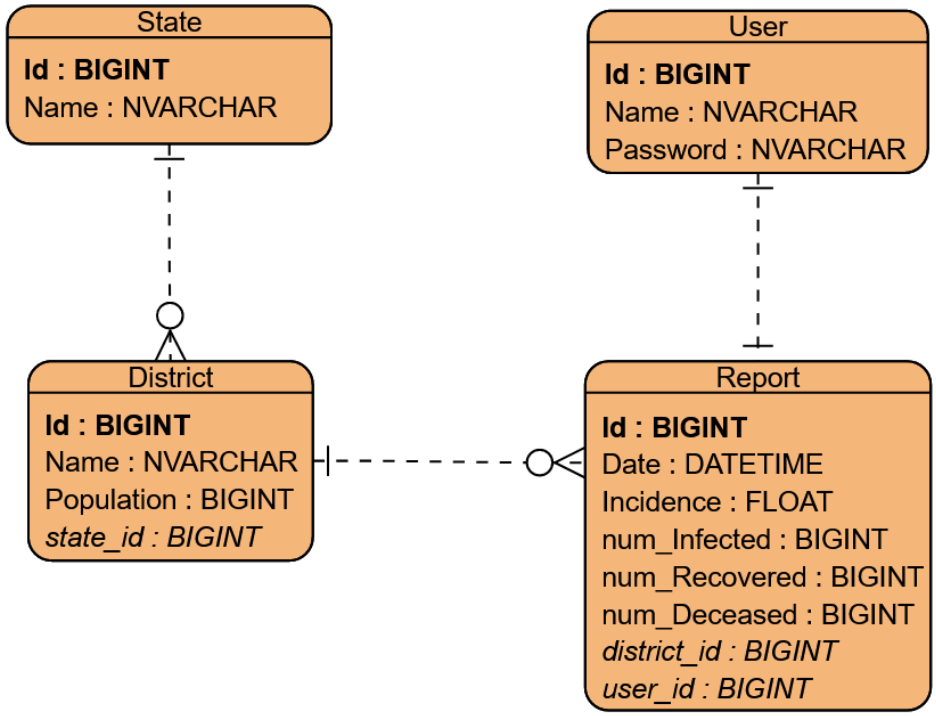
1.1 Domäne

Die Domäne bzw. die Daten sehen in meinem Fall so aus:



Ich habe also alle relevanten Daten in den Klassen gekapselt. Im Zuge meines Designs habe ich mich dazu entschieden nicht nur die IDs der Fremd-Entitäten zu speichern, sondern vollständige Referenzen auf die entsprechenden Klassen, um zu jeder Zeit vollständige Informationen zu haben.

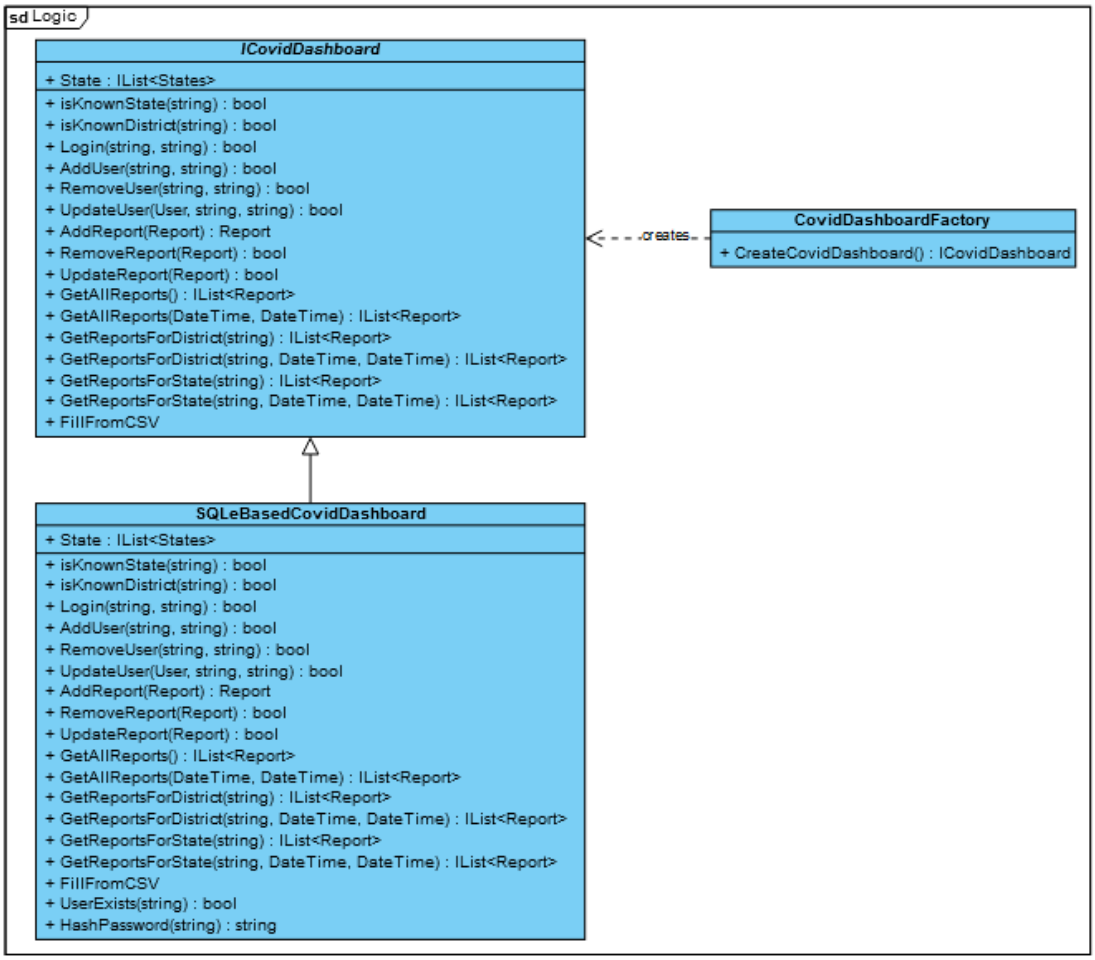
Aus diesen Klassen ergibt sich auch folgendes ER-Modell:



Anfangs dachte ich daran für District einen Composite-Key zu verwenden, was sich aber mit der Lösung aus dem Unterricht nicht gut vereinbaren lasst, da es wünschenswert ist, dass beim Einfügen bzw. Updaten die ID des Elements retourniert wird um dieses lokal zu vervollständigen bzw. eine Rückmeldung zu erhalten, ob die Datenbankbefehle erfolgreich waren oder nicht. Deswegen habe ich schlussendlich die gesamte GKZ (die indirekt das Bundesland enthält) als Key definiert.

1.2 Logic

Die zweite Komponente „Logic“ sieht wie folgt aus:



Wie im Unterricht wurde eine entsprechende Factory und ein entsprechendes Interface für alle gekapselten Funktionalitäten erstellt.

Ich würde zwar in der DAL-Komponente alle Datenbankoperationen für sämtliche Objekte in der Domäne implementieren, jedoch ist es für den Anwendungsfall den ich mir vorstelle nicht relevant, dass die DML-Operationen für Bundesländer und Bezirke nach außen hin verwendbar sind, da sich die Bundesländer und Bezirke voraussichtlich nicht ändern werden.

Deswegen ist es nur möglich den Datenbestand von Usern und Reports zu verändern. Zusätzlich dazu werden noch Funktionen und Properties zur Verfügung gestellt um leicht die Districts/States zu erhalten, was später bei WPF vorteilhaft sein wird. Die wirklichen Daten (Reports) wurden in verschiedenen Varianten selektiert um eine Filterung auf Bezirk oder Bundesland, sowie auf ein gewisses Zeitfenster zu ermöglichen.

Zusätzlich dazu wurden in der Generalisierung die Methoden UserExists und HashPassword angefügt, da ich die Passwörter sicher speichern will und ich mir vorstellen kann, dass es relevant ist, ob ein User (mit bestimmtem Namen) bereits existiert. Außerdem wurde eine Funktion vorgesehen, welche ein gegebenes CSV ausliest und dessen Daten in die Datenbank schreibt.

1.3 Kleine Änderungen

Ich habe mich, konträr zur Übung, dazu entschieden, dass ich die JOINS wirklich in den Commands/Selects selbst verwirklichen will und nicht nachträglich im Code die Einträge zusammenstöpseln muss. Daraus ergibt sich, dass sämtliche Read-Methoden für District und Report auch in der spezialisierten Klasse genauer ausgeführt werden müssen, da sich die select-statements nicht mehr in AbstractDao zusammenfassen lassen.

Als Lösung dafür würde ich eine dynamische Bindung implementieren, indem ich die generell formulierten Methoden in AbstractDao als virtual kennzeichne und dann in den relevanten Subklassen überschreibe.

2. Quellcode

Der Quellcode wurde aufgrund der schieren Menge an Dateien nicht direkt ins PDF kopiert, weshalb ich hier auf das Visual Studio Projekt verweise.

3. Testfälle

Ähnlich zum Quellcode werden auch diese im Detail im Code durchgegangen.

Die Testfälle, welche alle Funktionen des IcovidDashboard Interfaces abdecken, waren leider teils (nicht vermeidbar) abhängig voneinander, aber diesen Fakt habe ich versucht zu reduzieren, soweit es geht.

Anbei noch ein Screenshot des erfolgreichen Durchlaufs: