PLF in Programmieren und Software Engineering

> Klasse: 5AKIF
> Datum: MI, 26. November 2025
> Arbeitszeit: 3 UE

Wichtiger Hinweis vor Arbeitsbeginn

Auf dem Laufwerk *Z* finden Sie die Datei *Plf5akif.7z*. Klicken Sie mit der rechten Maustaste auf die Datei und wählen Sie *Weitere Optionen - 7-Zip* und *Extract here*. Gehen Sie dann in den Ordner *Plf5akif* und starten Sie die Datei *start_solution.cmd*. Diese Datei lädt zuerst alle Dependencies aus dem Internet und startet dann die sln Datei in diesem Ordner. Sie müssen das Programm nicht abgeben, denn Sie arbeiten direkt am Netzlaufwerk.

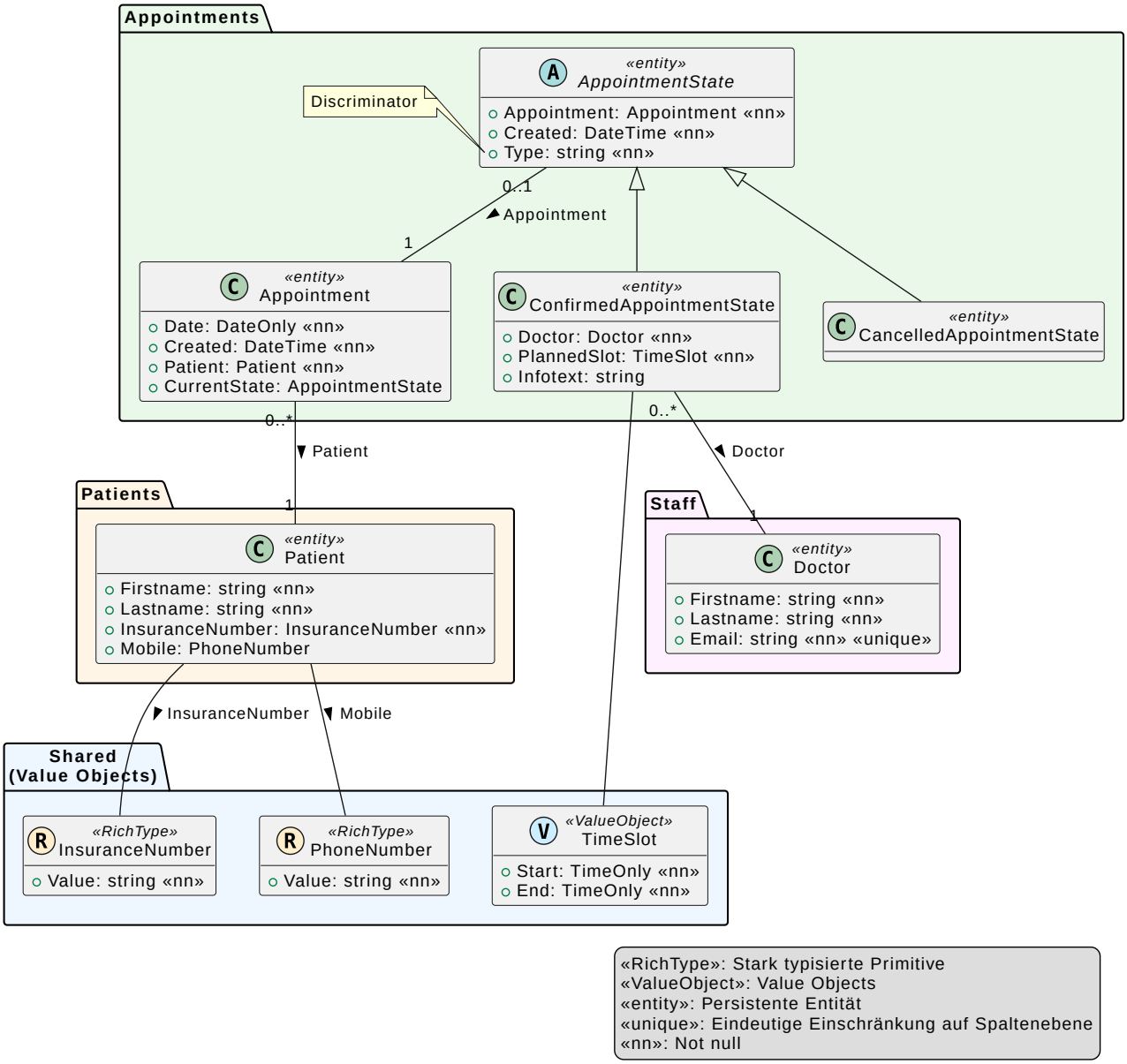
Füllen Sie die Datei *README.md* in *Plf5akif/README.md* mit Ihren Daten (Klasse, Name und Accountname) aus. Sie sehen die Datei in Visual Studio unter *Solution Items* nach dem Öffnen der Solution. **Falls Sie dies nicht machen, kann Ihre Arbeit nicht zugeordnet und daher nicht bewertet werden!**

Während der Prüfung ist der Internetzugriff gesperrt. Führen Sie daher niemals *Build - Rebuild Solution* aus, denn dadurch werden die lokalen Pakete gelöscht. Arbeiten Sie immer mit *Build - Build Solution* (F6).

Aufgabe: Object Relation Mapping

Für eine Arztpraxis soll ein Buchungssystem für Termine entwickelt werden. Patienten sollen einen Termin (Appointment) buchen können. Der Arzt (Doctor) kann den Termin bestätigen oder ablehnen. Wird der Termin bestätigt, hat das Appointment den Status *ConfirmedAppointmentState*. Wird der Termin abgesagt, hat das Appointment den Status *CandelledAppointmentState*.

Das Domain Model hat folgendes Aussehen:



Arbeitsauftrag

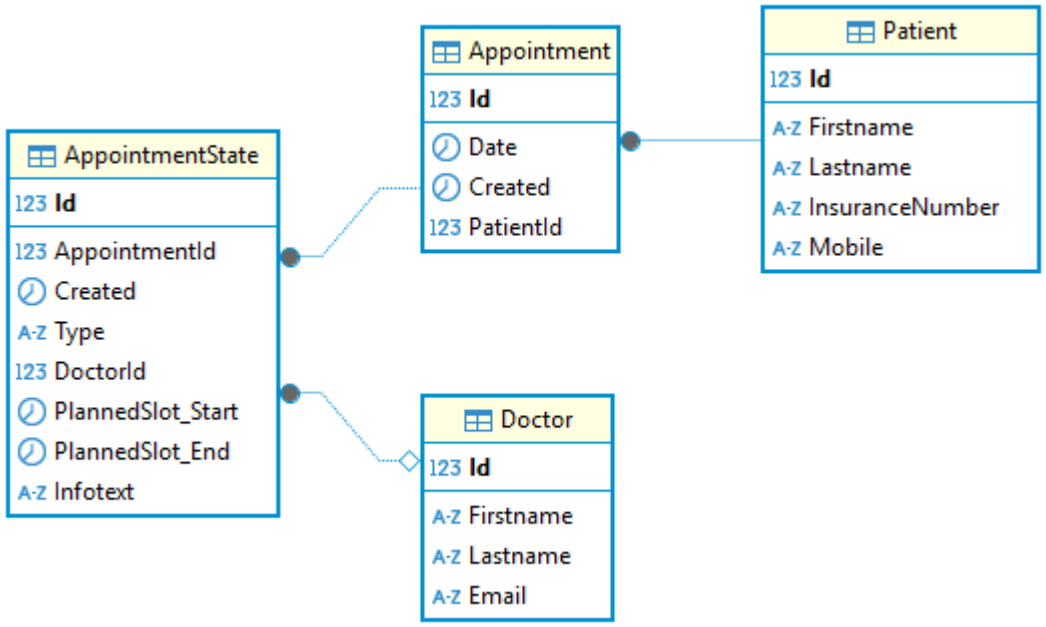
Erstellung der Modelklassen

Implementieren Sie das dargestellte Diagramm als EF Core Modelklassen. Im Projekt *Plf5a-kif.Application* befinden sich leere Klassen sowie die Klasse *AppointmentContext*, die Sie nutzen sollen. Beachten Sie bei der Umsetzung folgende Punkte:

- Legen Sie nötige Konstruktoren an. Ein *public* Konstruktor soll alle im Modell enthaltenen Properties initialisieren. Ergänzen Sie die für EF Core nötigen *protected* Konstruktoren.
- Beachten Sie Attribute Constraints wie *not null* (`<<nn>>`).
- Verwenden sie eigene primary keys mit dem Namen *Id* (autoincrement), außer im Modell ist mit *PK* explizit ein Schlüssel mit `<<PK>>` angegeben.
- Die Foreign Keys werden nach der Convention Propertyname + Name des PK generiert. Dies ist bei der Zuweisung des FKs der dependent Entities wichtig.

- Die generierten Tabellennamen sollen in Einzahl erzeugt werden (*AppointmentState*, *Appointment*, *Patient* und *Doctor*).
- Das Attribut *Email* in *Doctor* soll unique sein.
- Es sollen nicht zwei Appointments für den selben Patienten am selben Tag angelegt werden können. Stellen Sie dies durch ein unique constraint über diese 2 Spalten sicher.
- Das Attribut *PlannedSlot* soll in *ConfirmedAppointmentState* als *value object* definiert werden.
- Der Discriminator in *AppointmentState* soll in *Type* geschrieben werden und den Wert *Confirmed* für ein AppointmentState des Typs *ConfirmedAppointmentState* und *Cancelled* für den Typ *CancelledAppointmentState* haben.
- Das Attribut *InsuranceNumber* soll in *Patient* als rich type definiert und als String in der Datenbank gespeichert werden.
- Der rich type *InsuranceNumber* soll prüfen, ob der String 10 Stellen lang ist. Verwenden Sie dafür das Property *Length*. Falls nicht, werfen Sie mit `throw new AppointmentException("Invalid InsuranceNumber");` eine Exception.
- Das Attribut *Mobile* soll in *Patient* als rich type definiert und als String gespeichert werden. Beachten Sie, dass das Attribut auch null sein kann.

Das durch den OR Mapper erzeugte Datenbankschema soll so aussehen:



Sie können den vorgegebenen Test *T00_SchemaTest* in *GradingTests* verwenden, um das Gesamtmodell zu prüfen. Läuft dieser Test erfolgreich durch, sind Sie positiv.

Verfassen von Tests

In der Klasse *AppointmentContextTests* im Projekt *Plf5akif.Test* sollen Testmethoden verfasst werden, die die Richtigkeit der Konfiguration des OR Mappers beweisen sollen.

- *AddPatientTest* zeigt, dass Sie einen Patienten in die Datenbank einfügen können.
- *InsuranceNummerNotValidTest* zeigt, dass die Validierung in *InsuranceNumber* funktioniert.

Sie müssen im Test nur eine Instanz dieser Klasse anlegen und das Fehlerverhalten prüfen.

- *AddAppointmentWithStateConfirmedTest* zeigt, dass Sie ein Appointment mit dem Status *ConfirmedAppointmentState* erzeugen und in der Datenbank speichern können.
- *DoctorEmailThrowsDbUpdateExceptionIfNotUniqueTest* zeigt, dass beim Einfügen eines zweiten Doctor Datensatzes mit selber E-Mail eine DbUpdateException geworfen wird.

Bewertung

Um eine positive Beurteilung erreichen zu können, muss das Programm kompilieren und die registrierten Klassen müssen mit der EF Core Funktion `EnsureCreated()` eine Datenbank ohne Laufzeitfehler erzeugen.

Aufgabe (36 Punkte in Summe)	**Ges**
Das Entity Doctor wird korrekt in der erzeugten Datenbank abgebildet.	2
Das Entity Patient wird korrekt in der erzeugten Datenbank abgebildet.	2
Das Entity Appointment wird korrekt in der erzeugten Datenbank abgebildet.	2
Das Entity AppointmentState wird korrekt in der erzeugten Datenbank abgebildet.	2
Das Entity ConfirmedAppointmentState wird korrekt in der erzeugten Datenbank abgebildet.	2
Das Entity CancelledAppointmentState wird korrekt in der erzeugten Datenbank abgebildet.	2
Das Attribut Doctor.Email ist unique	1
Die Attribute Appointment.Date und Appointment.PatientId sind unique.	2
Der Discriminator wurde in AppointmentState.Type gemappt.	1
Der Discriminator in AppointmentState.Type hat die korrekten Werte.	2
Das Attribut ConfirmedAppointmentState.TimeSlot ist als value object definiert.	2
Das Attribut Patient.Mobile ist als rich type definiert.	2
Das Attribut Patient.InsuranceNumber ist als rich type definiert.	2
Der Test AddPatientTest hat den richtigen Aufbau.	2
Der Test AddPatientTest läuft durch.	1
Der Test InsuranceNummerNotValidTest hat den richtigen Aufbau.	2
Der Test InsuranceNummerNotValidTest läuft durch.	1
Der Test AddAppointmentWithStateConfirmedTest hat den richtigen Aufbau.	2
Der Test AddAppointmentWithStateConfirmedTest läuft durch.	1
Der Test DoctorEmailThrowsDbUpdateExceptionIfNotUniqueTest hat den richtigen Aufbau.	2
Der Test DoctorEmailThrowsDbUpdateExceptionIfNotUniqueTest läuft durch.	1

36 - 32 Punkte: Sehr gut, 31 - 28 Punkte: Gut, 27 - 23 Punkte: Befriedigend, 22 - 19 Punkte: Genügend, 18 - 0 Punkte: Nicht genügend.