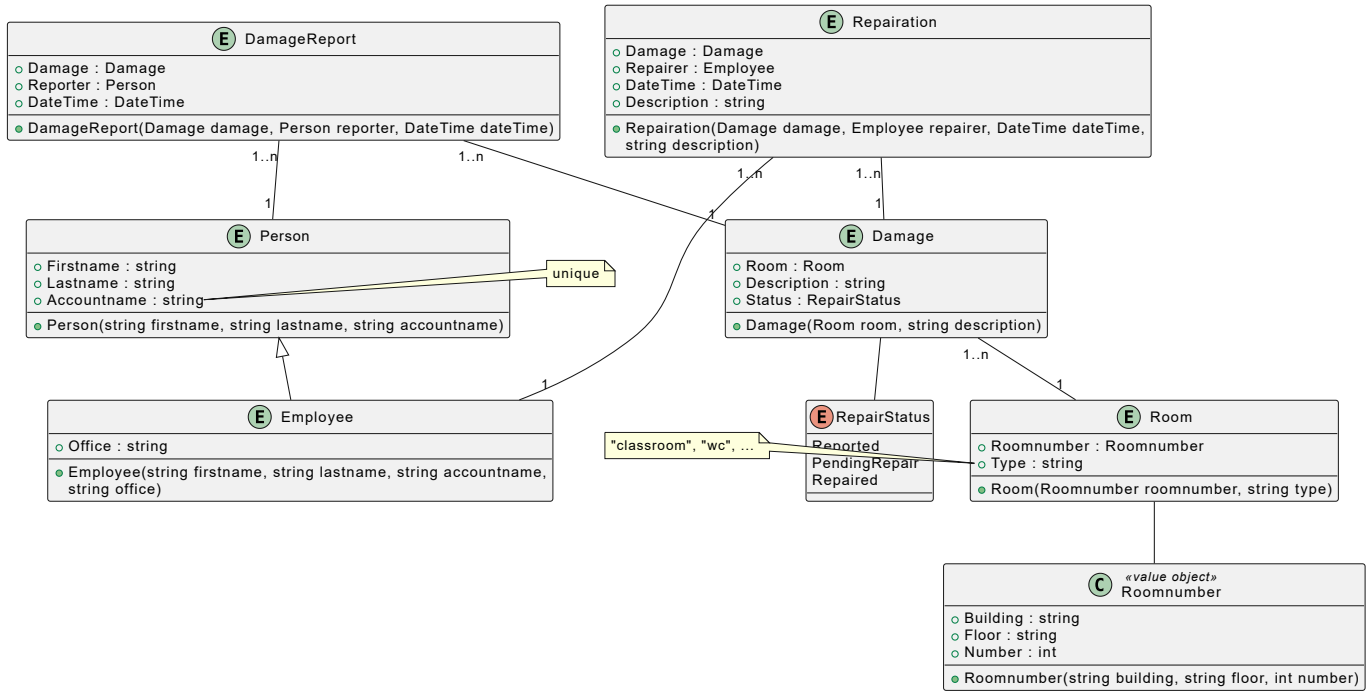
Erstellen von EF Core Modelklassen

Ein System zum Melden von Schäden

In den letzten Jahren sind Beschädigungen im Haus (leider) immer häufiger ein Thema. Es soll daher ein Meldesystem entwickelt werden, mit dessen Hilfe Studierende, Lehrende und Mitarbeiter/innen der Schule Schäden, die sie entdecken, melden können. Die Applikation soll so aufgebaut werden, dass zuerst ein Raum ausgewählt wird. Danach werden alle Schäden, die schon gemeldet wurden, aufgelistet. Es kann entweder ein neuer Schaden erfasst oder ein bestehender Schaden neu gemeldet werden. So werden Doppelmeldungen vermieden. Deswegen hat ein Schaden im vorgesehenen Modell auch mehrere Meldungen. Die Hausverwaltung hat Einsicht und kann Reparaturen veranlassen. Eine Reparatur besteht manchmal aus mehreren Schritten, da oft Firmen beauftragt werden müssen. Diese Schritte müssen in der Applikation auch dokumentiert werden können.

Modell



Beschreibung der Klassen

Roomnumber (value object)

Die Raumnummern der Schule haben die Form C2.14, etc. Damit die Teile für Abfragen getrennt gespeichert werden, wird ein value object verwendet.

- *Building:* Gebäude ("A", "B", "C" oder "D").
- *Floor:* Stockwerk (1-5), "E", "U" oder "H".
- *Number:* Nummer des Raumes.

Room

Bildet einen Raum in der Schule ab.

- *Roomnumber:* Die gesamte Raumnummer, Referenz auf das *Roomnumber* Objekt.

- *Type:* Frei belegbarer String wie "Stammklasse", "WC", ...

Person

Alle in der Schule beschäftigten Personen wie Studierende, Lehrende und Mitarbeiter/innen.

- *Firstname:* Vorname.
- *Lastname:* Nachname.
- *Accountname:* Accountname im Active Directory der Schule.

Employee (erbt von Person)

- *Office:* Mitarbeiter/innen haben zusätzlich ein Büro. Hier wird die Raumnummer als String gespeichert.

RepairStatus (enum)

Der Status eines gemeldeten Schadens wird als enum gespeichert. Er kann mehrere Zustände haben:

- *Reported*: Der Schaden wurde gemeldet, die Hausverwaltung hat aber noch nicht reagiert.
- *PendingRepair*: Die Hausverwaltung hat den Schaden bestätigt und Reparaturmaßnahmen eingeleitet.
- *Repaired*: Der Schaden wurde behoben.

Damage

Wird ein Schaden gemeldet, wird ein Datensatz in der Tabelle *Damage* angelegt.

- *Room:* In welchem Raum wurde die Beschädigung festgestellt?
- *Description:* Eine kurze Beschreibung (Tisch verschmutzt, Steckdose funktioniert nicht, etc.).
- *Status:* Der Status der Reparatur wie in der enum *RepairStatus* angegeben.

DamageReport

Da ein Schaden mehreren Studierenden auffällt, kann er mehrmals gemeldet werden. So entsteht ein Überblick, wie lange der Schaden schon besteht.

- *Damage:* Verweis auf den Schaden.
- *Reporter:* Die Person, die die Beobachtung macht.
- *DateTime:* Der Zeitpunkt der Beobachtung.

Repairation

Für einen Schaden können mehrere Reparaturschritte notwendig sein. Oft müssen Teile erst bestellt bzw. externe Firmen beauftragt werden. Daher wird in Repairation ein "Log" geführt, mit dessen Hilfe die Hausverwaltung die einzelnen Schritte dokumentieren kann.

- *Damage:* Verweis auf den Schaden.
- *Repairer:* Mitarbeiter/in (Employee), der den Schaden bearbeitet.
- *DateTime:* Datum und Uhrzeit des Eintrages.
- *Description:* Beschreibung (z. B. "Firma X beauftragt", "Schaden behoben", ...)

Arbeitsauftrag

Erstellung der Modelklassen

Im Projekt *SPG_Fachtheorie.Aufgabe1* befinden sich im Ordner *Model* leere Klassendefinitionen. Bilden Sie jede Klasse gemäß dem UML Diagramm ab, sodass EF Core diese persistieren kann. Beachten Sie folgendes:

- Wählen Sie selbst notwendige Primary keys.
- Definieren Sie Stringfelder mit vernünftigen Maximallängen (z. B. 255 Zeichen für Namen, etc.).
- Roomnumber ist ein *value object*. Stellen Sie durch Ihre Definition sicher, dass kein Mapping dieser Klasse in eine eigene Datenbanktabelle durchgeführt wird.
- Das Feld *Accountname* in *Person* muss unique sein. Stellen Sie dies durch eine geeignete Konfiguration sicher.
- Legen Sie Konstruktoren mit allen erforderlichen Feldern an. Erstellen Sie die für EF Core notwendigen default Konstruktoren als *protected*.
- Wird ein neuer Schaden angelegt, hat dieser den Status *Reported*, der im Konstruktor gesetzt wird.
- Das Feld *Status* in *Damage* ist ein enum Feld. Speichern Sie dieses Feld als String in der Datenbank. Stellen Sie dies durch geeignete Konfiguraton sicher.
- Implementieren Sie die Vererbung korrekt, sodass eine (1) Tabelle *Person* entsteht.
- Legen Sie die erforderlichen DB Sets im Datenbankcontext an.

Verfassen von Tests

Im Projekt *SPG_Fachtheorie.Aufgabe1.Test* ist in *Aufgabe1Test.cs* der Test *CreateDatabaseTest* vorgegeben. Er muss erfolgreich durchlaufen und die Datenbank erzeugen. Sie können die erzeugte Datenbank in *C:/Scratch/Aufgabe1_Test/Debug/net8.0/damages.db* in SQLite Studio öffnen.

Implementieren Sie folgende Tests selbst, indem Sie die minimalen Daten in die (leere) Datenbank schreiben. Leeren Sie immer vor dem *Assert* die nachverfolgten Objekte mittels *db.ChangeTracker.Clear()*.

- Der Test *AddEmployeeSuccessTest* beweist, dass Sie einen Mitarbeiter (Employee) in die Datenbank einfügen können. Prüfen Sie im Assert, ob der angegebene Accountname auch korrekt gespeichert wurde.
- Der Test *AddDamageWithReportSuccessTest* beweist, dass Sie einen Schaden samt Meldung (Report) anlegen können. Legen Sie hierfür einen Schaden (Damage) samt Raum, Employee und Person an und fügen Sie eine Meldung ein. Stellen Sie im Assert sicher, dass der gespeicherte Schaden eine Meldung hat.
- Der Test *AddRepairationSuccessTest* beweist, dass Sie eine Reparatur für einen Schaden anlegen können. Legen Sie dafür einen Schaden samt der notwendigen Daten an und weisen einen Reparatureintrag zu. Prüfen Sie im Assert, ob die Beschreibung der Reparatur in der Datenbank vorhanden ist.

Bewertung (24P, 38.7% der Gesamtpunkte)

Jedes der folgenden Kriterien wird mit 1 Punkt bewertet.

- Die Stringfelder verwenden sinnvolle Längenbegrenzungen.
- Die Klasse *Roomnumber* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *Room* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *Room* besitzt ein korrekt konfiguriertes value object *Address*.
- Die Klasse *Room* wurde korrekt im DbContext registriert.
- Die Klasse *Person* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.

- Die Klasse *Person* wurde korrekt im DbContext registriert.
- Das Feld *Accountname* in *Person* ist als unique definiert.
- Die Klasse *Employee* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *Employee* wurde korrekt im DbContext registriert.
- Die Klasse *Employee* erbt korrekt von der Klasse *Person*.
- Die Klasse *Damage* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *Damage* wurde korrekt im DbContext registriert.
- Das Feld *RepairStatus* in *Damage* wird in der Datenbank als String gespeichert.
- Die Klasse *DamageReport* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *DamageReport* wurde korrekt im DbContext registriert.
- Die Klasse *Repairation* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *Repairation* wurde korrekt im DbContext registriert.
- Der Test *AddEmployeeSuccessTest* ist korrekt aufgebaut.
- Der Test *AddEmployeeSuccessTest* läuft erfolgreich durch.
- Der Test *AddDamageWithReportSuccessTest* ist korrekt aufgebaut.
- Der Test *AddDamageWithReportSuccessTest* läuft erfolgreich durch.
- Der Test *AddRepairationSuccessTest* ist korrekt aufgebaut.
- Der Test *AddRepairationSuccessTest* läuft erfolgreich durch.