Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw School of
Management and Law

eUmzug Kanton Bern

Aufgabe 6: Persistenz-Layer VeKa und GWR

Modul Geschäftsprozessintegration im BSc Wirtschaftsinformatik

Autor
Björn Scheppler
scep@zhaw.ch

Herausgeber
Zürcher Hochschule für Angewandte Wissenschaften
School of Management and Law
Institut für Wirtschaftsinformatik (IWI)
Stadthausstrasse 14
CH-8041 Winterthur
www.zhaw.ch/iwi

Vorliegende Version
1.0.0 (31.10.2018)

Änderungshistorie

Version (Datum)	Wer	Was
1.0.0 (31.10.2018)	scep	Initiale Fassung

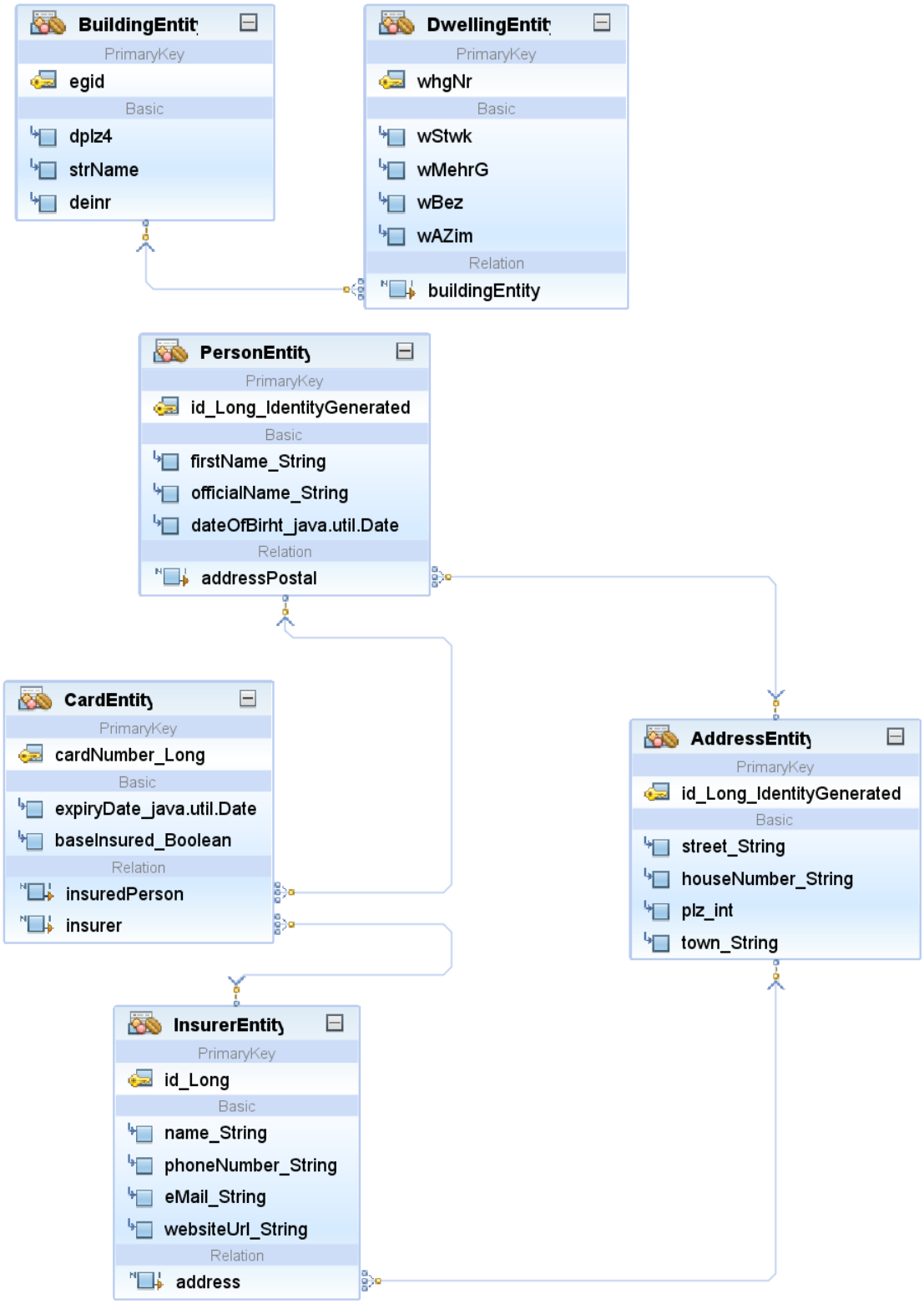
Hinweis: Aus Gründen der Lesbarkeit wurde im Text stets die männliche Form gewählt.

1 Zielsetzung

1. Am Beispiel von den Umsystemen VeKa und GWR habt ihr «nackte» **SpringBoot**-Applikationen als **Maven**-Projekte Schritt-für-Schritt aufgebaut.
2. Am Beispiel dieser Umsysteme habt ihr zudem gefestigt, wie man mit einer relationalen Datenbank kommuniziert ohne SQL-Befehle schreiben zu müssen, sondern stattdessen **ORM** verwendet, in unserem Fall Hibernate als Implementierung von JPA im Spring-Framework.
3. Beim **VeKa**-Umsystem werden die **Stammdaten** von Versicherern, Versicherten, Versichertenkarten und Adressen verwaltet. Für diese sind **alle Persistierungs-Komponenten** vorhanden von den Entities über Repositories, Datenbank und Testdaten bis hin zu einer testenden CommandLineRunner-Klasse.
4. Dasselbe gilt analog für das **GWR** (Gebäude- und Wohnungsregister) mit den Stammdaten zu Gebäuden und Wohnungen.

2 Vorgeschlagenes Vorgehen

1. Besucht die **Kleinklasse** in der Semesterwoche 7, in welcher Björn mit euch die Starthilfe durchgeht. Damit habt ihr …
    a. … im Idealfall bereits **Punkt 1 der Kriterien** aus Kapitel 4.1 erfüllt sowie
    b. … gemeinsam mit dem Stoff von Peter aus der Grossklasse dieser und letzter Woche das **praktische Rüstzeug**, für die selbständig zu erarbeitenden Teile.
2. Beginnt mit dem Erstellen der **Maven-Projekte**:
    a. Das Vorgehen wurde von Björn am Beispiel des VeKa-Umsystems in der Starthilfe in der Kleinklasse gezeigt.
    b. Geht absolut analog vor für das GWR. Wo jeweils *VeKa* steht, ersetzt dies durch *Gwr*, *veka* durch *gwr*. Der Server-Port ist 8090. Die Datei mit den Testdaten befindet sich hier.
3. Nun zu den **Entitäten**:
    a. Erstellt die Entitäten gemäss den unten stehenden **Modellen** für GWR (oben) und VeKa (unten).
        i. Für das **GWR** findet ihr in dieser Dokumentation des GWR jeweils die Angaben zu den Feldern im Modell, welche Einschränkungen sie haben, wenn ihr den Abschnitt «Codierung» und «Technische Spezifikationen» lest. Einschränkungen, welche über die bereits gelernten Annotationen (@Size, @Min, usw.) hinausgehen, sind nicht zu berücksichtigen (z.B. *Alle Zahlen und Buchstaben, [´], [-] und [.]*). Um die entsprechenden Seiten im Dokument überhaupt zu finden, sucht im PDF nach den Feldbezeichnungen in Grossbuchstaben (z.B. STRNAME für strName).
        ii. Das **VeKa**-Modell ist bewusst sehr prototypisch gehalten. So sind z.B. bei der *street* keine weiteren Einschränkungen vorhanden im Unterschied etwa zu *strName* beim GWR.



4. Erstellt nun diejenigen **Repositories** mit allenfalls eigenen **Methoden**, soweit dies erforderlich ist, damit die im nächsten Schritt erstellten Methoden funktionieren.

5. Erstellt nun in einem controller-Package **Controller**-Klassen mit folgenden Methoden, welche später im Semester via REST (VeKa) respektive SOAP (GWR) für die Umzugsplattform zur Verfügung stehen. Diese Methoden greifen auf die vorher erstellten Repositories zu.
    a. **GWR**:
        i. **Adress-Existenz prüfen**: Gibt true zurück, falls EIN Gebäude (BuildingEntity) zu einer Adresse existiert, ansonsten false. Eine Adresse setzt sich zusammen aus strName, deinr und dplz4.
        ii. **Wohnungen eines Gebäudes**: Gibt eine Liste von Wohnungen (DwellingEntity) zu einem der Methode übergebenem Gebäude zurück.
    b. **VeKa**:
        i. **Karten-Informationen zu einer Kartennummer**: Gibt ein Karten-Objekt (CardEntity) zu einer übergebenen Kartennummer zurück.
6. Startet eure Projekte, meldet euch in der H2-Konsole an und versucht, die **Testdaten** (initalData.sql) einzufügen. Klappt es nicht, passt entweder die SQL-Statements an eure Entities an oder die Entities an die SQL-Statements und versucht es erneut.
7. Fakultativ, aber empfohlen: Ruft in einem **TempCommandLineRunner** eure Methoden aus den Controller-Klassen aus und gebt die zurückgegebenen Objekte in der Konsole aus oder prüft über Debug-Modus, ob diese den erwarteten Inhalt haben für verschiedene Testdaten (z.B. unterschiedliche Kartennummern).

3 Abgabe

1. Die inhaltliche Abgabe ist ein Commit und Push in den zwei Github-Repositories (VeKa und GWR) der Gruppe.
2. Die formale Abgabe erfolgt auf Moodle hier durch EINE Person im Team.
3. Dies erfolgt spätestens bis zum im Betreff der verlinkten Aufgabe aufgeführten Zeit.
4. Die Abgabe besteht darin, die Ids der von uns zu korrigierenden Commits als Text einzufügen wie folgt und auf «Abgabe einreichen» zu klicken

Veka: [Commit ID]

GWR: [Commit ID]

4 Bewertungskriterien

4.1 Kriterien

Maximal können **10 Punkte** erreicht werden, die sich auf die folgenden **Kriterien** aufteilen:

1. Teile aus **Kleinklassen-Unterricht-Starthilfe** (*Zusätzliche Repositories und Projekte*) korrekt implementiert (1 Punkte)
2. Funktionierende und korrekte **Entitäten** gemäss den Anforderungen aus dem Klassendiagramm (3 Punkte)
3. Funktionierende **Repositories** mit allfälligen eigenen Methoden. Nur Repositories und

Methoden erstellen, die für die Controller-Methoden gemäss nächstem Punkt erforderlich sind (2 Punkte)

4. Funktionierende **Controller**-Klassen und -Methoden gemäss den Anforderungen in Punkt 5 des vorgeschlagenen Vorgehens (2 Punkte)
5. **Eleganz** und **Robustheit** des Codes: Gemeint ist, dass man «funktionierenden» Code erstellen kann, der aber sehr umständlich zum Ziel kommt (nicht elegant) und/oder Code, der mit Fehler auf zu erwartende Ausnahmefälle reagiert (z.B. NullPointerException, wenn kein Eintrag in einer Datenbank gefunden werden kann). Natürlich sind wir prototypisch unterwegs und haben nicht den Anspruch, alles abzufangen. Wenn ihr euch am Code des Personenregisters und der Musterlösung für die Aufgabe 5 orientiert, seid ihr auf gutem Weg (1 Punkt)

4.2 Aufwandreduktion 4er-Gruppen

In den Entitäten BuildingEntity und DwellingEntity kann darauf verzichtet werden, die Attribute genauso umzusetzen, wie sie in der GWR-Dokumentation aufgeführt sind (das Dokument kann daher ignoriert werden). Stattdessen muss nur sichergestellt werden, dass Zahlen als int und Zeichenfolgen als String umgesetzt werden. Welche Attribute als Zahlen und welche als Zeichenfolgen zu interpretieren sind, lässt sich aus initialData.sql ableiten.

4.3 0 Punkte

Keine Abgabe innerhalb der Abgabefrist auf Moodle und Github.

4.4 Punkteabzug

Für die folgenden formalen Fehler und fehlenden/falschen Dokumentationen gibt es die Abzüge auf die erreichte Punktzahl gemäss Kapitel 4.1. Diese Abzüge werden aufsummiert, wobei eine Gruppe nicht weniger als 0 Punkte erreichen kann.

1. **Keine fristgerechte Abgabe auf Moodle**, aber auf Github eine Lösung gepusht: 4 Punkte
2. **Auf Moodle formal eine falsche Abgabe** (z.B. keine vorhandene Commit-ID, etwas anderes als eine Commit-ID, usw.): 2 Punkte
3. Java-Klassen in falsch(en) (bezeichneten) **Packages**: 1 Punkt
4. Java-Klassen, Methoden und Variablen **falsch benannt** (Klassen anders als vorgegeben [soweit vorgegeben] sowie Gross-/Kleinschreibung): maximal 2 Punkte
5. Klassen und Methoden ohne aussagekräftige Java Documentation (**JavaDoc**). Enthält eine Klasse nur eine Methode, so reicht eine Java Documentation der Klasse: maximal 4 Punkte
6. Jedes Java-Statement (mit Semikolon beendet) sowie if-then-else- oder for-Blöcke enthlaten einen oder mehrere zutreffende, selbst formulierte, deutschsprachige **Zeilenkommentare**. Davon ausgenommen sind Standard-Getter- & Setter-Methoden sowie wenn direkt untereinander mehrfach dasselbe gemacht wird (z.B. Auslesen von vier Prozessvariablen in lokale Variablen). Verletzung dieser Regel: maximal 4 Punkte.

5 Feedback

Spätestens am 22.11.2018 10:00 ist die **Musterlösung** unterhalb der Aufgabenstellung auf Moodle freigeschaltet.

Spätestens am 5.12.2018 20:00 sieht diejenige Person, welche die Aufgabe eingereicht hat, die erreichte Punktzahl pro Kriterium und insgesamt inklusive allfälliger Kommentare, wenn sie erneut auf die **Moodle-Aufgabe** klickt.

6 Fortgeschrittene Anforderungen (optional)

6.1 Mögliche zu implementierende Anforderungen

6.1.1 Liste aller erfassten Versicherer (einfach)

Ausgangslage: Es wäre hilfreich, wenn man auf einer Webseite eine Liste aller im VeKa-System erfassten Versicherer ausgeben könnte.

Anforderung: Eine weitere Controller-Methode sowie vermutlich im Hintergrund auch eine dazugehörige Repository-Methode gibt eine List<InsurerEntity> zurück, welche alphabetisch nach Name der Versicherung aufsteigend sortiert ist.

6.1.2 Versicherung suchen (einfach)

Ausgangslage: Standardmässig im JpaRepository steht eine findById-Methode zur Verfügung. Aber in der Regel kennt man nicht die Id (BAG-Nummer), sondern die Bezeichnung der Versicherung oder einen Teil davon (z.B. *Atupri*) und möchte daher über diese suchen können.

Anforderung: Eine weitere Controller-Methode sowie vermutlich im Hintergrund auch eine dazugehörige Repository-Methode gibt eine List<InsurerEntity> zurück, welche alle Versicherer enthält, welche einen übergebenen Suchtext im Namen enthalten, unabhängig von Gross-/Kleinschreibung.

6.2 Abgabe

Die Regeln in Kapitel 5.2 der Hauptaufgabestellung einhalten. Falls die Abgabe nicht erst am 24.12.2018, sondern gleichzeitig mit den Basis-Anforderungen erfolgt, dann in der Abgabe auf Moodle (siehe Kapitel 3 oben) nicht bloss die Commit Id angeben, sondern auf einer zweiten Zeile auch der Hinweis: «Fortgeschrittene Anforderungen implementiert».