

Probeklausur in POS

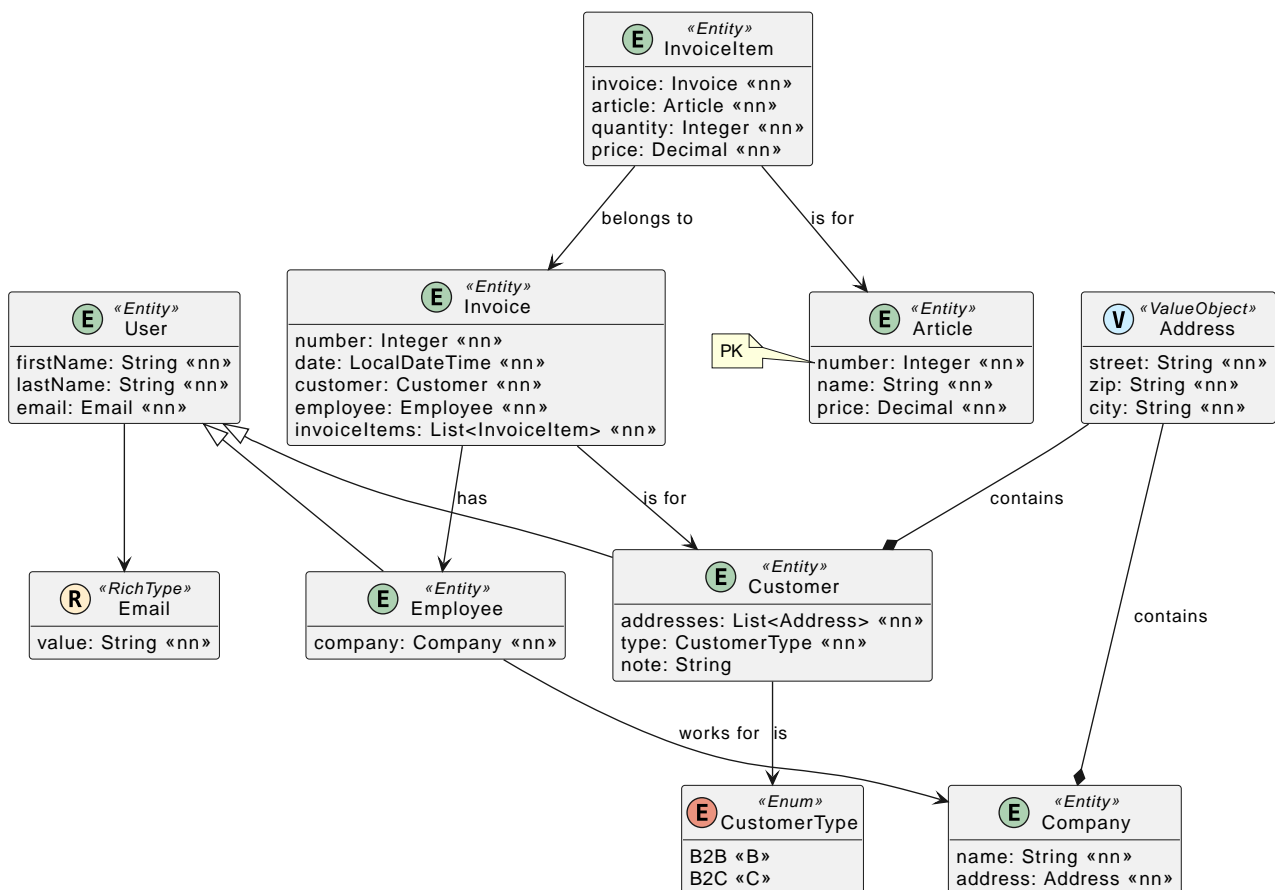
Klasse: 5CHIF

Datum: 19. März 2025

Arbeitszeit: 5 UE

Teilaufgabe 1: Object Relation Mapping und Database Queries

Bilden Sie das dargestellte Modell entsprechend Ihres gewählten Technologie Stacks sauber ab. Beachten Sie dabei die geforderten Anforderungen an einzelne Modellklassen.



Arbeitsauftrag

Erstellung der Datenbankzugriffe

Erstellen Sie die üblichen Datenbank Zugriffsobjekte Ihres gewählten Technologie Stacks. Fügen Sie - wenn es die Abfragen erleichtert - auch zusätzliche Navigations hinzu. Beachten Sie Attribute Constraints wie not null (nn). Setzen Sie folgende Abfragen um:

- ¥ Listen Sie alle Artikel auf, die innerhalb eines bestimmten Zeitraumes gekauft wurden. Geben Sie Artikelnummer, Artikelname, Kaufdatum und den Vor- und Zunamen des Kunden aus.
- ¥ Listen Sie alle Verkäufe auf, die ein bestimmter Mitarbeiter getätigt hat. Geben Sie die Rechnungsnummer, das Rechnungsdatum, den Vor- und Zunamen des Kunden und den Gesamtbetrag aus.

Verfassen von Tests

Stellen Sie durch automatisierte Tests sicher, dass:

- ¥ Sie alle Elemente des Modells richtig gemapped haben und diese in der DB gespeichert werden.
- ¥ Sie notwendige Umschlüsselungen (Richtypes, Enumerations) in beide Richtungen richtig implementiert haben.
- ¥ Die geforderten Abfragen die richtigen Ergebnisse liefern und nur die geforderten (!) Daten liefern.

Teilaufgabe 2: Services und Unittests

Es wird Ihnen zu Testzwecken eine Datenzugriffs-Schnittstelle sowie die zur Implementierung notwendigen Datenklassen zur Verfügung gestellt. Sie sollen kein OR-Mapping implementieren, sondern eine isolierte Umsetzung des Services anstreben.

Das nachfolgende Modell zeigt eine Umsetzung eines Scooter Sharing Services. Kunden können zwischen 2 Abrechnungsmethoden wählen:

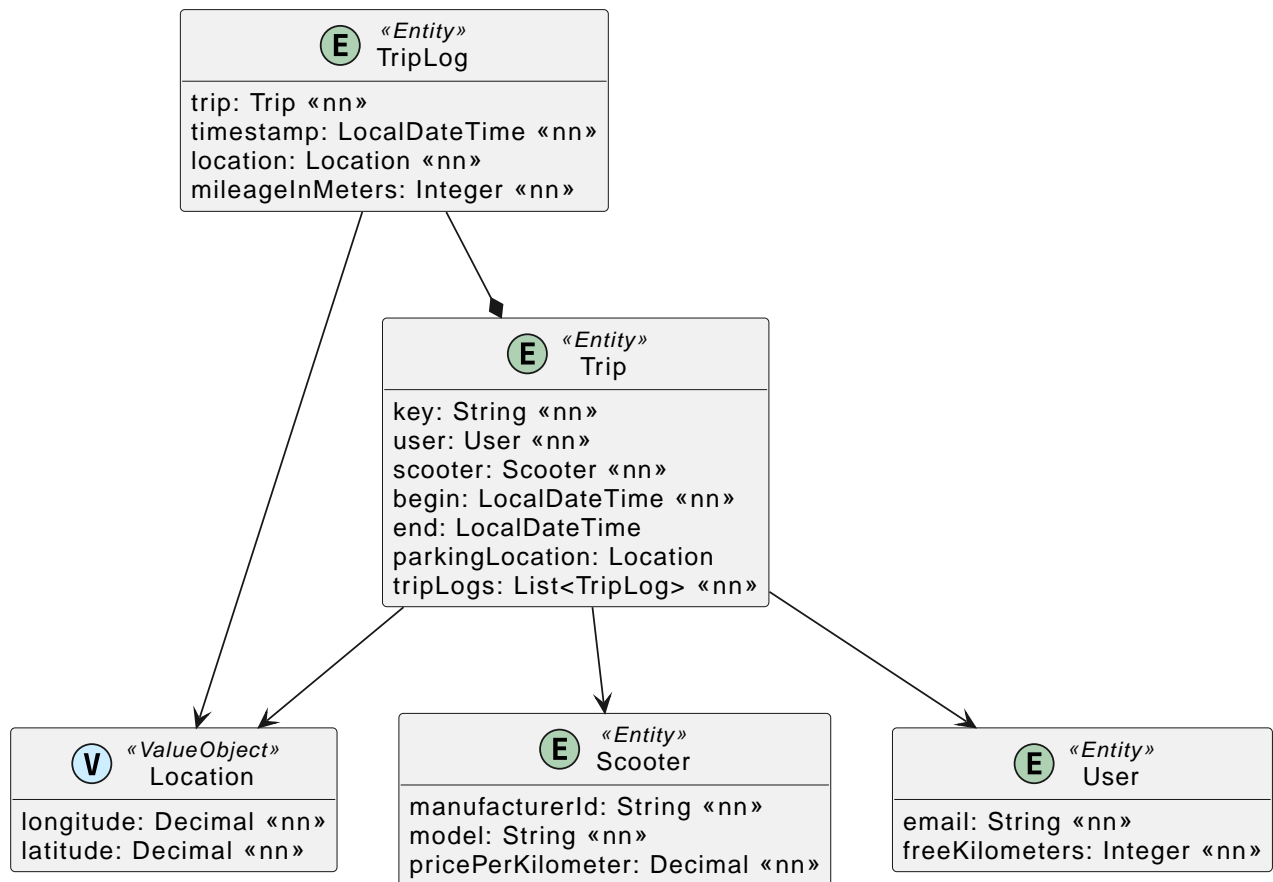
- ¥ *Pay as you go*: Der Kunde zahlt pro gefahrenen Kilometer.
- ¥ *Included Kilometers*: Der Kunde zahlt einen monatlichen Beitrag und erhält eine bestimmte Anzahl an Freikilometern. Die freien Kilometer gelten pro Fahrt. Hat ein Kunde z. B. 2 Freikilometer, so wird bei Fahrt A, die 3 km lang ist, 1 km berechnet. Bei Fahrt B, die 4 km lang ist, werden 2 km berechnet.

Beim Entsperren mit der App wird ein *Trip* gestartet. Der integrierte Tracker des Scooters übermittelt in regelmäßigen Abständen die Position und den Kilometerstand als *TripLog*. Dadurch ist eine Abrechnung auch dann möglich, wenn kein durchgehendes GPS Signal verfügbar ist (z. B. Unterführungen, etc). Die Länge einer Fahrt (*Trip*) ist dann die Differenz zwischen niedrigsten und höchsten Kilometerstand der *TripLogs* dieser Fahrt.

Die Preise für den gefahrenen Kilometer variieren je nach Modell und sind in der *Scooter* Klasse hinterlegt. Scooter mit Sitz und hoher Reichweite haben z. B. einen höheren Preis pro Kilometer als Scooter ohne Sitz.

Wird ein Scooter abgestellt, wird der *Trip* beendet und die Felder *end* und *parkingLocation* werden

den gesetzt. So ist erkennbar, welche Trips noch laufen und welche bereits beendet sind.



Arbeitsauftrag

Implementierung von Servicemethoden

Schreiben Sie eine Methode zur Abrechnung von Kunden (Usern). Geben Sie eine Map mit den User IDs und den Abrechnungsinformationen pro Trip zurück. Jedes Element der Liste ist ein Trip und enthält:

¥ Die gefahrenen Kilometer des Trips.

¥ Der Preis des Trips.

Für die Preiskalkulation sollen die Felder *pricePerKilometer* und *freeKilometers* berücksichtigt werden. Es sollen nur abgeschlossene Trips berücksichtigt werden, d. h. alle Trips, wo das Feld *end* gesetzt ist.

Key	Value
User ID 1	Liste der gefahrenen Kilometer und des Preises
User ID 2	Liste der gefahrenen Kilometer und des Preises
User ID 3	Liste der gefahrenen Kilometer und des Preises

Key	Value
É	É

Verfassen von Tests

Stellen Sie durch automatisierte Tests sicher, dass:

- ¥ Sie die Statistik richtig berechnen und
- ¥ die Datenstrukturen richtig sortiert bereitstellen.

Teilaufgabe 3: REST(ful) API

Fr das vorige Modell der Scooter Sharing App soll eine RESTful API implementiert werden. Der Key in Trip soll vorab auf Validitt geprft werden. Ein Trip Key kann mit folgendem regulren Ausdruck geprft werden: `^TR[0-9]+$`

Arbeitsauftrag

Implementieren Sie die folgenden REST API Routen. Testen Sie entsprechend des angegebenen Antwortverhaltens (Status-Code, Responses).

GET /trips/{key}?includeLog=true

Diese REST API Route soll einen bestimmten Trip mitsamt der Logeintrge retournieren. Wird der optionale Parameter *includeLog* mit dem Wert *true* mitgegeben (*false* ist der Default-Wert), soll die Antwort die Logeintrge enthalten. Ist der parameter *false*, soll ein leeres Array zurck! gegeben werden.

Table 1. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
200	Ein <i>TripDto</i> , welches optional <i>TripLogDtos</i> enthlt
400	Fr einen fehlerhaften Key (inkl. RFC-9457 ProblemDetail im Body)
404	Fr einen unbekannten Key (inkl. RFC-9457 ProblemDetail im Body)

Schematische Response (*TripDto*):

```
{
  "key": string,
  "userEmail": string,
  "scooterManufacturerId": string,
  "begin": string,
  "end": string|null,
```

```

{
  "logs": [
    {
      "timestamp": string,
      "longitude": number,
      "latitude": number
      "mileageInMeters": number
    }
  ]
}

```

PATCH /trips/{key}

Diese REST API Route soll die Daten am Ende eines Trips aktualisieren. Dabei werden die Felder *End* und *ParkingLocation* auf die entsprechenden Werte gesetzt.

Table 2. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
200	Ein <i>TripDto</i> Objekt.
400	F�r einen fehlerhaften Key. (inkl. RFC-9457 ProblemDetail im Body)
400	Wenn ein Trip, der bereits beendet ist, modifiziert werden soll. (inkl. RFC-9457 ProblemDetail im Body)
404	F�r einen unbekannten Key. (inkl. RFC-9457 ProblemDetail im Body)

Payload (*UpdateTripCommand*):

```

{
  "end": string,
  "longitude": number,
  "latitude": number
}

```

Schematische Response (*TripDto*):

Siehe GET Route.

Bewertung

Die Bewertung erfolgt nach folgenden Kriterien:

Teilaufgabe 1 (34 Punkte, 50% der Gesamtpunkte)

- ¥ (1 P) Die Klasse *Company* ist korrekt als Entity abgebildet.
- ¥ (1 P) Die Klasse *Company* besitzt ein korrekt konfiguriertes value object *Address*.
- ¥ (1 P) Die Klasse *Address* wurde korrekt als value object abgebildet.
- ¥ (1 P) Die Klasse *User* ist korrekt als Entity abgebildet.
- ¥ (1 P) Die Klasse *Employee* ist korrekt als Entity abgebildet.
- ¥ (1 P) Die Vererbung von *User* zu *Employee* ist korrekt abgebildet.
- ¥ (1 P) Die Klasse *Customer* ist korrekt als Entity abgebildet.
- ¥ (1 P) Die Vererbung von *User* zu *Customer* ist korrekt abgebildet.
- ¥ (1 P) Die Klasse *Customer* besitzt ein konfiguriertes value object *Address*.
- ¥ (1 P) Die Klasse *Invoice* ist korrekt als Entity abgebildet.
- ¥ (1 P) Die Klasse *Article* ist korrekt als Entity abgebildet.
- ¥ (1 P) Die Klasse *InvoiceItem* ist korrekt als Entity abgebildet.
- ¥ (2 P) Der rich type *Email* wird korrekt abgebildet.
- ¥ (2 P) Die enum *CustomerType* wird mit einem Converter korrekt gespeichert.
- ¥ (2 P) Der Unittest *PersistRichTypesSuccessTest* hat einen korrekten Aufbau und läuft durch.
- ¥ (2 P) Der Unittest *PersistEnumSuccessTest* hat einen korrekten Aufbau und läuft durch.
- ¥ (2 P) Der Unittest *PersistValueObjectInCompanySuccessTest* hat einen korrekten Aufbau und läuft durch.
- ¥ (2 P) Der Unittest *PersistValueObjectInCustomerSuccessTest* hat einen korrekten Aufbau und läuft durch.
- ¥ (2 P) Der Unittest *PersistInvoiceItemSuccessTest* hat einen korrekten Aufbau und läuft durch.
- ¥ (2 P) Die Methode *GetArticleWithSales* liefert ein korrektes Ergebnis und verwendet LINQ.
- ¥ (2 P) Der Unittest *GetArticleWithSalesInfoSuccessTest* hat einen korrekten Aufbau und läuft durch.
- ¥ (2 P) Die Methode *GetEmployeeWithSales* liefert ein korrektes Ergebnis und verwendet LINQ.
- ¥ (2 P) Der Unittest *GetEmployeeWithSalesSuccessTest* hat einen korrekten Aufbau und läuft durch.

Teilaufgabe 2 (12 Punkte, 16.7% der Gesamtpunkte)

- ¥ (1 P) Die Methode *CalculateTripInfos* filtert die Trips korrekt (nur beendete Trips).
- ¥ (1 P) Die Methode *CalculateTripInfos* liefert ein Dictionary mit einem Eintrag pro User.

- ¥ (1 P) Die Methode *CalculateTripInfos* liefert ein Dictionary mit einer Liste von TripInfo Objekten pro User.
- ¥ (1 P) Die Methode *CalculateTripInfos* liefert ein Dictionary mit einem TripInfo Objekt pro Trip.
- ¥ (1 P) Die Methode *CalculateTripInfos* berechnet die gefahrenen Kilometer aufgrund des Minimal- und Maximalwertes des Properties *MileageInMeters* im TripLog korrekt.
- ¥ (1 P) Die Methode *CalculateTripInfos* ermittelt den Fahrpreis unter Berücksichtigung des Properties *PricePerKilometer* korrekt.
- ¥ (1 P) Die Methode *CalculateTripInfos* ermittelt den Fahrpreis unter Berücksichtigung des Properties *FreeKilometers* korrekt.
- ¥ (5 P) Der vorgegebene Unittest *CalculateTripInfos_ReturnsCorrectResult* läuft durch.

Teilaufgabe 3 (24 Punkte, 33.3% der Gesamtpunkte)

- ¥ (1 P) Der GET Endpunkt wurde richtig annotiert und ist in ASP.NET Core verfügbar.
- ¥ (1 P) Der GET Endpunkt filtert die Daten korrekt.
- ¥ (1 P) Der GET Endpunkt liefert das korrekte Verhalten im Fall eines ungültigen Keys.
- ¥ (1 P) Der GET Endpunkt liefert das korrekte Verhalten im Fall eines nicht vorhandenen Keys.
- ¥ (2 P) Der GET Endpunkt verwendet den Query Parameter *includeLog* korrekt.
- ¥ (1 P) Die DTO Klasse *TripDto* wurde korrekt implementiert.
- ¥ (1 P) Die DTO Klasse *TripLogDto* wurde korrekt implementiert.
- ¥ (2 P) Der GET Endpunkt liefert das korrekte Ergebnis im Erfolgsfall.
- ¥ (3 P) Der Integration Test beweist die korrekte Implementierung des GET Endpunktes.
- ¥ (1 P) Der PATCH Endpunkt wurde richtig annotiert und ist in ASP.NET Core verfügbar.
- ¥ (1 P) Der PATCH Endpunkt liefert das korrekte Verhalten im Fall eines ungültigen Keys.
- ¥ (1 P) Der PATCH Endpunkt liefert das korrekte Verhalten im Fall eines nicht vorhandenen Keys.
- ¥ (1 P) Der PATCH Endpunkt liefert das korrekte Verhalten für einen Trip, der bereits beendet ist.
- ¥ (1 P) Der PATCH Endpunkt aktualisiert die Daten korrekt in der Datenbank.
- ¥ (1 P) Die Command Klasse *UpdateTripCommand* wurde korrekt implementiert.
- ¥ (2 P) Der PATCH Endpunkt liefert das korrekte Ergebnis im Erfolgsfall.
- ¥ (3 P) Der Integration Test beweist die korrekte Implementierung des PATCH Endpunktes.

Beurteilung:

72 - 64 Punkte: Sehr gut (1),

63 - 55 Punkte: Gut (2),

54 - 46 Punkte: Befriedigend (3),

45 - 37 Punkte: Gen gend (4),

36 - 0 Punkte: Nicht gen gend (5)

Viel Erfolg!