Klausurprüfung in Programmieren und Software Engineering

> Klassen: 6AAIF, 6BAIF, 6CAIF, 6AKIF, 6BKIF
> Datum: DO, 15. Jänner 2026
> Arbeitszeit: 300 Minuten

Generelle Hinweise zur Bearbeitung

Die Arbeitszeit für die Bearbeitung der gestellten Aufgaben beträgt 5 Stunden (300 Minuten). Die 3 Teilaufgaben sind unabhängig voneinander zu bearbeiten, Sie können sich die Zeit frei einteilen. Wir empfehlen jedoch eine maximale Bearbeitungszeit von 2 Stunden für Aufgabe 1, 1 Stunde für Aufgabe 2 und 2 Stunden für Aufgabe 3. Bei den jeweiligen Aufgaben sehen Sie den Punkteschlüssel. Für eine Einrechnung der Jahresnote sind mindestens 30% der Gesamtpunkte zu erreichen.

Hilfsmittel

In der Datei *P:/SPG_Fachtheorie/SPG_Fachtheorie.sln* befindet sich das Musterprojekt, in dem Sie Ihren Programmcode hineinschreiben. Im Labor steht Visual Studio 2022 mit der .NET Core Version 8 zur Verfügung. Da das Projekt auf dem Netzlaufwerk liegt, muss am Ende nicht extra auf ein Abgabelaufwerk kopiert werden. Beenden Sie am Ende Ihrer Arbeit alle Programme und lassen Sie den PC eingeschaltet.

Als erlaubte Hilfsmittel befinden sich im Ordner **R:\exams** bereitgestellte Unterlagen. Dies ist ein implementiertes Projekt ohne Kommentare aus dem Unterricht, wo Sie die Parameter von benötigten Frameworkmethoden nachsehen können.

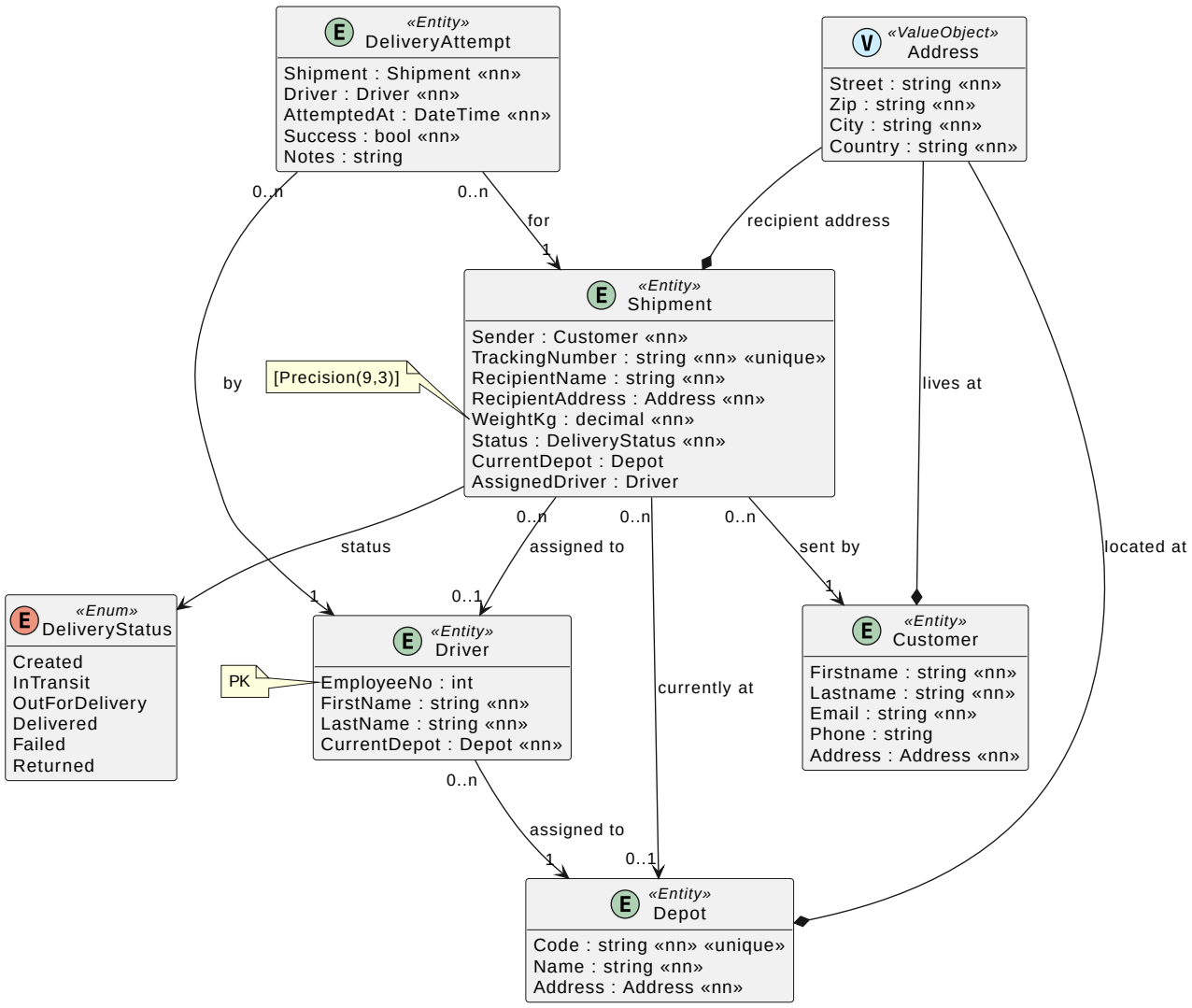
Wichtiger Hinweis vor Arbeitsbeginn

Füllen Sie die Datei *README.md* in *SPG_Fachtheorie/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!**

Teilaufgabe 1: Object Relation Mapping

Der Paketdienst **FastShip** benötigt ein System zur Verwaltung von Sendungen vom Eingang im Depot bis zur Zustellung. Kund:innen geben Pakete im Depot auf; Fahrer:innen holen Sendungen ab und versuchen sie zuzustellen. Jeder Zustellversuch wird protokolliert.

Das folgende Diagramm beschreibt das zu implementierende Domänenmodell:



Domänenbeschreibung (kurz)

- *Depot* verwaltet die Standorte der Paketlager (eindeutiger Code) und deren Adresse.
- *Customer* (Absender:in) mit Kontaktinformationen und Adresse.
- *Driver* ist der Fahrer, der im Unternehmen beschäftigt ist.
- *Shipment* ist der Versandauftrag. Er besitzt die Empfängerinformation (*RecipientName* und *RecipientAddress*), das Gewicht der Sendung und eine *TrackingNumber* (eindeutig). Zu Beginn ist noch kein Depot und kein Driver zugewiesen. Deswegen sind die Felder nullable.

Sie werden erst später gesetzt, wenn diese Information verfügbar ist. Der Status wird über *DeliveryStatus* geführt.
- *DeliveryAttempt* protokolliert jeden Zustellversuch (Zeitpunkt, Erfolg, Notiz) für eine Sendung durch den Fahrer.

Arbeitsauftrag

Erstellung der Modelklassen

Implementieren Sie das dargestellte Diagramm als EF Core Modelklassen. Im Projekt *SPG_Fachtheorie_Aufgabe1* befinden sich leere Klassen sowie die Klasse *FastShipContext*, die Sie nutzen sollen. Beachten Sie bei der Umsetzung folgende Punkte:

- Legen Sie sinnvolle *Konstruktoren* an:
  - Ein *public*-Konstruktor initialisiert alle Properties.
  - Fügen Sie die für EF Core nötigen protected default Konstruktoren hinzu.
- *Required/Optional* gemäß Diagramm (<<*nn*>> = required).
- *Datentypen*:
  - *Strings* sollen die maximale Länge von *255* Zeichen haben.
  - *Shipment.WeightKg* mit *[Precision(9, 3)]*.
- *Value Object*:
  - *Address* ist als *Owned Type* zu konfigurieren und wird in *Depot.Address*, *Customer.Address* und *Shipment.RecipientAddress* verwendet.
- *Enums*:
  - *DeliveryStatus* in *Shipment* wird als *String* gespeichert.
- *Primary Keys*:
  - Verwenden Sie einen Primärschlüssel mit dem Namen *Id* (Autoincrement) für jene Entities, wo kein PK im Diagramm angegeben wird.
- *Unique Constraints / Indizes*:
  - *Depot.Code* (*unique*)
  - *Shipment.TrackingNumber* (*unique*)

Verfassen von Tests

In der Klasse *Aufgabe1Test* im Projekt *test/SPG_Fachtheorie.Aufgabe1.Test* sollen Testmethoden verfasst werden, die die Richtigkeit der Konfiguration des OR Mappers beweisen sollen.

- *PersistDeliveryAttemptTest* weist nach, dass eine Zustellung (Shipment) samt Zustellversuch (DeliveryAttempt) gespeichert werden kann.
- *EnsureDepotCodeIsUniqueTest* weist nach, dass *Depot.Code* nicht doppelt gespeichert werden

kann.

- *EnsureTrackingNumberIsUniqueTest* weist nach, dass *Shipment.TrackingNumber* nicht doppelt gespeichert werden kann.

Sie können die vorgegebenen Tests in *test/SPG_Fachtheorie.Aufgabe1.Test/Aufgabe1MasterTests* verwenden, um Ihre Konfiguration bei der Bearbeitung der Aufgabenstellung zu prüfen.

Teilaufgabe 2: Abfragen und Servicemethoden

Das folgende Modell beschreibt ein vereinfachtes System für einen Online-Shop, in dem Kund:innen Produkte aus verschiedenen Kategorien vorbestellen können. Jede Vorbestellung kann mehrere Positionen enthalten, die jeweils auf Produkte aus dem Katalog verweisen. Die Daten bilden die Grundlage für Abfragen, REST-Endpunkte und einfache Geschäftslogik im Service-Layer.

Übersicht der Entitäten

Category

Beschreibt eine Produktkategorie (z. B. Elektronik, Haushalt, Bücher). Eine Kategorie kann mehreren Produkten zugeordnet sein.

Product

Repräsentiert ein Produkt, das in einer Kategorie geführt wird. Enthält Name, Beschreibung und Preis. Produkte können in Vorbestellungen referenziert werden.

Customer

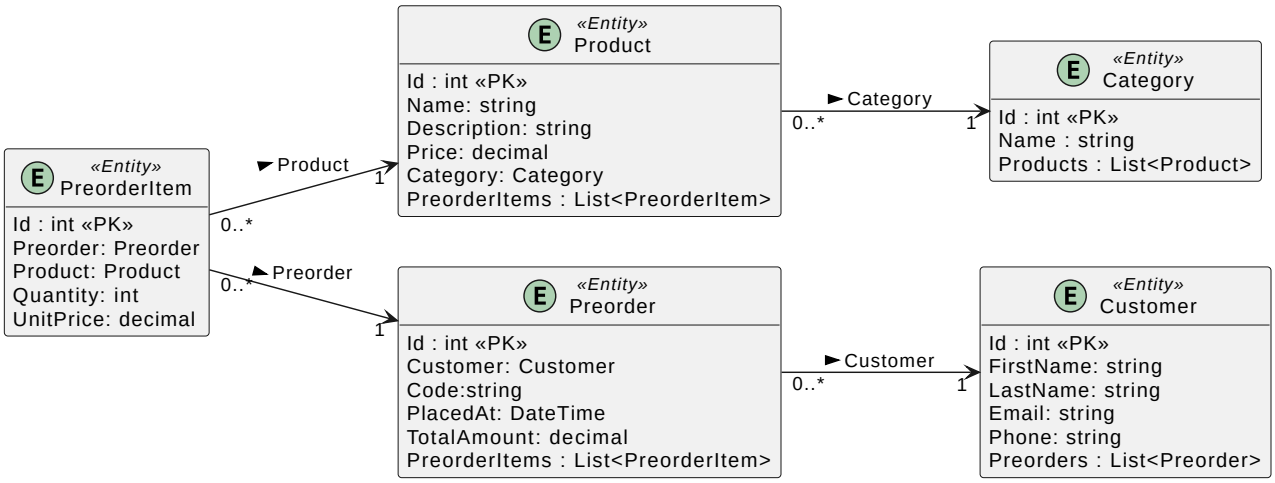
Enthält Stammdaten der Kund:innen (Name, E-Mail, Telefonnummer). Ein Kunde kann mehrere Vorbestellungen aufgeben.

Preorder

Stellt eine Vorbestellung eines Kunden dar. Enthält den Bestellcode, das Datum der Bestellung und den Gesamtbetrag. Eine Vorbestellung kann mehrere Positionen enthalten.

PreorderItem

Repräsentiert eine einzelne Position innerhalb einer Vorbestellung. Verweist auf ein Produkt, enthält die bestellte Menge und den zum Zeitpunkt der Bestellung gültigen Einzelpreis.



Arbeitsauftrag

In der Datei *src/SPG_Fachtheorie.Aufgabe2.Base/Infrastructure/OnlineStoreContext.cs* steht ein vollständig implementierter DbContext zur Verfügung. Das Model ist bereits in *src/SPG_Fachtheorie.Aufgabe2.Base/Model* implementiert. **Sie müssen keine Implementierung des Modelles vornehmen, sondern es in Ihren Servicemethoden verwenden.**

Implementierung der Servicemethoden

Führen Sie Ihre Implementierungen in *src/SPG_Fachtheorie.Aufgabe2/Services/OnlineStoreService.cs* durch. Verwenden Sie den über Dependency Injection bereitgestellten *OnlineStoreContext*.

public List<CategoryWithCountDto> GetCategoriesWithProductCounts()

Geben Sie alle Produktkategorien (*Category*) samt der Anzahl der darin enthaltenen Produkte zurück. Als Rückgabetyp steht Ihnen der folgende Record zur Verfügung.

```
public record CategoryWithCountDto(string CategoryName, int ProductCount);
```

public List<PreorderDto> GetPreordersOfCustomer(int customerId)

Geben Sie alle Vorbestellungen (*Preorder*) eines Kunden zurück. Als Rückgabetyp steht Ihnen der folgende Record zur Verfügung. Das Property *CustomerName* soll als Stringverknüpfung mit dem Muster *{FirstName} {LastName}* gebildet werden.

```
public record PreorderDto(int CustomerId, string CustomerName, string PreorderCode, DateTime
PreorderPlacedAt, decimal PreorderTotalAmount);
```

public List<ProductWithRevenueDto> GetRevenueOfProduct(int productId)

Gibt den Umsatz eines Produktes zurück. Um den Umsatz zu berechnen, summieren Sie *PreorderItem.Quantity * PreorderItem.UnitPrice* aller Vorbestellungen (*PreorderItem*) des Produktes. Als Rückgabetyp steht Ihnen der folgende Record zur Verfügung.

```
public record ProductWithRevenueDto(int ProductId, string ProductName, decimal Revenue)
```

public Preorder AddPreorder(int customerId, List<CustomerProductPreorder> productPreorders)

Legen Sie eine Vorbestellung (*Preorder*) samt der PreorderItems in der Datenbank an. Im Parameter des Typs *List<CustomerProductPreorder>* bekommen Sie folgende Informationen:

```
public record CustomerProductPreorder(int ProductId, int Quantity)
```

Für das Feld *PreorderItem.UnitPrice* übernehmen Sie den Wert in *Product.Price*. Für das Feld *TotalAmount* summieren Sie *Quantity * UnitPrice* jedes vorbestellten Produktes.

Beachten Sie folgende Randbedingungen:

- Existiert die *customerId* nicht in der Datenbank, wird eine *OnlineStoreException* mit dem Text *"Invalid CustomerId."* geworfen.
- Ist die Liste *productPreorders* leer, wird eine *OnlineStoreException* mit dem Text *"Empty preorders."* geworfen.
- Wird eine Product ID in *productPreorders* nicht in der Datenbank gefunden, wird diese einfach ignoriert.

Projizieren Sie zuerst die übergebene Liste *productPreorders* in eine Liste von *PreorderItem*. Danach fügen Sie die Preorder ein und weisen mit *Preorder.PreorderItems.AddRange()* die Preorder Items hinzu.

Testen Ihrer Implementierung

Verwenden Sie die vorgegebenen Tests in *test/SPG_Fachtheorie.Aufgabe2.Test/Aufgabe2MasterTests*, um Ihre Implementierung bei der Bearbeitung zu prüfen.

Teilaufgabe 3: REST(ful) API

Für das vorige Modell des Online Stores soll eine RESTful API implementiert werden. Wenn Sie das Projekt in *src/SPG_Fachtheorie.Aufgabe3* starten, steht Ihnen unter der URL *http://localhost:5080/swagger/index.html* ein Endpoint Explorer zur Verfügung. Die Datenbank

beinhaltet Musterdaten, sodass Sie die Funktionalität Ihrer Controller damit testen können.

Arbeitsauftrag

Implementieren Sie die folgenden REST API Routen. In *src/SPG_Fachtheorie.Aufgabe3/Controllers/CustomersController.cs* steht dafür ein leerer Controller bereit. Über Dependency Injection wird ein mit Musterdaten gefüllter *OnlineStoreContext* bereitgestellt. Sie können die Abfragen direkt in den Controllermethoden implementieren. Führen Sie alle Abfragen nicht blockierend (mit *await* und *async*) durch. Achten Sie bei Fehlern auf eine RFC-9457 kompatible Antwort, indem Sie die Methode *Problem()* mit geeignetem Statuscode verwenden.

GET /customers

Diese REST API Route soll eine Liste von Kunden zurückliefern. Für die Rückgabe steht Ihnen folgender Record bereit:

```
public record CustomerDto(int Id, string FirstName, string LastName, string Email, string Phone);
```

Table 1. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
200	Eine Liste von *CustomerDto* Objekten mit allen Kunden.

GET /customers/preorder/{code}

Diese REST API Route soll eine Vorbestellung mit dem in der Adresse übergebenen Code zurückliefern. Der Code ist in *Preorder.Code* gespeichert. Für die Rückgabe steht Ihnen folgender Record bereit:

```
public record PreorderDto(DateTime PlacedAt, decimal TotalAmount, List<PreorderItemDto>
PreorderItems);
public record PreorderItemDto(string ProductName, int Quantity, decimal UnitPrice);
```

Der Code muss mindestens 5 Stellen lang sein. Stellen Sie dies durch Prüfung sicher und geben ohne Datenbankzugriff HTTP 400 (bad request) zurück, wenn der Code nicht mindestens 5 Stellen hat. Wenn der Code in der Datenbank nicht gefunden wurde, geben Sie HTTP 404 (not found) zurück.

Table 2. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
200	Ein *PreorderDto* Objekt mit den erforderlichen Daten.
400	Der Code hat nicht mindestens 5 Stellen.

HTTP Status	Bedingung
404	Der Code wurde nicht in der Datenbank gefunden.

POST /customers

Diese REST API Route soll einen neuen Kunden erstellen. Es steht Ihnen folgender Record bereit:

```
public record NewCustomerCmd(string FirstName, string LastName, string Email, string Phone);
```

Führen Sie vor dem Einfügen folgende Validierungen durch:

- FirstName hat mindestens 1 und maximal 255 Stellen.
- LastName hat mindestens 1 und maximal 255 Stellen.
- Die Email Adresse besitzt ein @ Zeichen.

Wird eine dieser Bedingungen verletzt, sollen Sie HTTP 400 (bad request) zurückliefern. Konnte der Kunde erstellt werden, liefern Sie HTTP 201 (created) mit der generierten Kunden ID (Property *Customer.Id*) zurück.

Table 3. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
201	Der Kunde konnte erstellt werden.
400	Ein Validierungskriterium wurde verletzt.

DELETE /customers/{id}

Dieser Endpunkt soll Kunden aus der Datenbank löschen. Die übergebene ID als Routingparameter ist der Wert in *Customer.Id*. Sie dürfen den Kunden allerdings nur löschen, wenn keine Preorders für diesen Kunden in der Datenbank vorhanden sind.

Table 4. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
204	Der Kunde konnte gelöscht werden.
400	Der Kunde hat angelegte Preorders und kann daher nicht gelöscht werden.
404	Der Kunde wurde nicht gefunden.

Verfassen von Integration Tests

In *test/SPG_Fachtheorie.Aufgabe3.Test/CustomersControllerTests.cs* sollen Integration Tests **für den DELETE Endpunkt** verfasst werden. Es soll jede Zeile in den Tabellen bei *Erwartete HTTP-Antworten* geprüft werden. Prüfen sie bei einer erfolgreichen Antwort auch, ob der Inhalt in der

Datenbank tatsächlich gelöscht wurde. Verwenden Sie dafür die in der *TestWebApplicationFactory* bereitgestellten Hilfsmethoden:

Methode	Beschreibung
InitializeDatabase	Erstellt eine leere Datenbank und fügt ggf. Werte ein.
QueryDatabase	Erlaubt das Abfragen der Datenbank.
GetHttpContent<T>	Führt einen GET Request durch und liefert das Ergebnis als Typ *T*.
DeleteHttpContent	Führt einen DELETE Request durch und liefert den Status Code zurück.

Wichtiger Hinweis nach Arbeitsende

Starten Sie am Ende Ihrer Arbeit - nach dem Schließen der IDE - das Skript *compile.cmd* im Ordner *SPG_Fachtheorie*. Es kontrolliert, ob Ihre Projekte kompiliert werden können. **Projekte, die nicht kompilieren, können nicht bewertet werden!**