2. Probeklausur in Programmieren und Software Engineering

> Klassen: 6AAIF, 6BAIF, 6CAIF, 6AKIF, 6BKIF
> Datum: FR, 20. Juni 2025
> Arbeitszeit: 5 UE

Generelle Hinweise zur Bearbeitung

Die Arbeitszeit für die Bearbeitung der gestellten Aufgaben beträgt 5 Unterrichtseinheiten. Die 3 Teilaufgaben sind unabhängig voneinander zu bearbeiten, Sie können sich die Zeit frei einteilen. Wir empfehlen jedoch eine maximale Bearbeitungszeit von 2 UE für Aufgabe 1, 1 UE für Aufgabe 2 und 2 UE für Aufgabe 3. Bei den jeweiligen Aufgaben sehen Sie den Punkteschlüssel.

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

Ein Online-Marktplatz soll entwickelt werden, auf dem Nutzer:innen gebrauchte Artikel verkaufen und kaufen können.

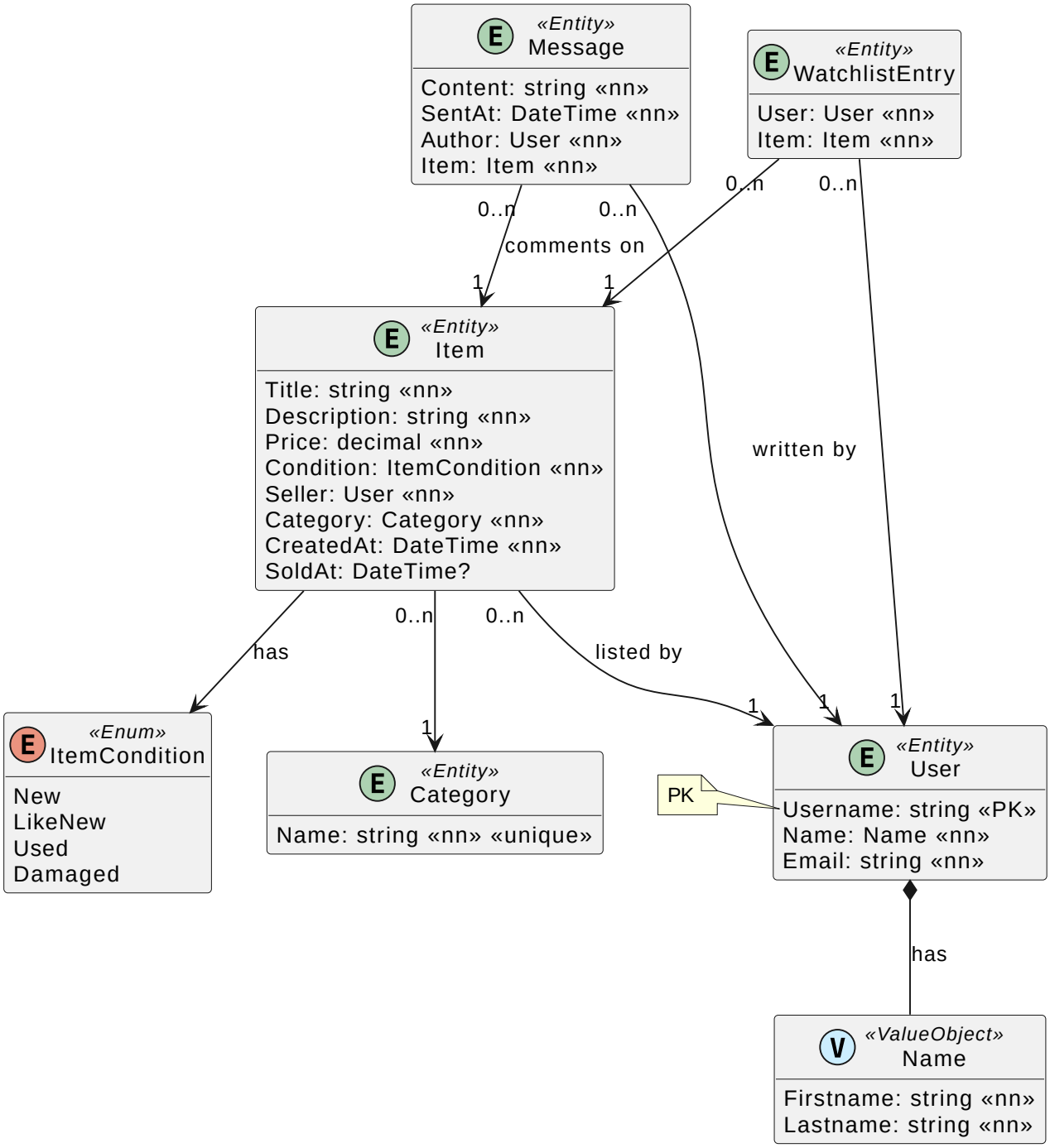
Jede Nutzer:in wird im Entity *User* erfasst. Dabei werden der eindeutige Benutzername (*User.Username*), der Name als Value Object (*User.Name*, bestehend aus *Name.Firstname* und *Name.Lastname*) sowie die E-Mail-Adresse (*User.Email*) gespeichert.

Artikel, die angeboten werden, sind im Entity *Item* hinterlegt. Ein Artikel verfügt über einen Titel (*Item.Title*), eine Beschreibung (*Item.Description*), einen Preis (*Item.Price*), einen Zustand (*Item.Condition*, basierend auf dem Enum *ItemCondition* mit den Werten *New*, *LikeNew*, *Used* und *Damaged*), ein Erstellungsdatum (*Item.CreatedAt*) sowie optional ein Verkaufsdatum (*Item.SoldAt*). Zusätzlich sind der:die Verkäufer:in (*Item.Seller*, referenziert auf *User*) und die zugehörige Kategorie (*Item.Category*) verknüpft.

Die möglichen Kategorien werden im Entity *Category* gepflegt. Jede Kategorie hat einen eindeutigen Namen (*Category.Name*), der systemweit einzigartig ist.

Zur Kommunikation können Nutzer:innen Nachrichten zu Artikeln verfassen. Diese werden im Entity *Message* gespeichert und enthalten den Nachrichtentext (*Message.Content*), den Versandzeitpunkt (*Message.SentAt*), sowie Referenzen zur verfassenden Person (*Message.Author*) und zum Artikel (*Message.Item*).

Nutzer:innen können interessante Artikel auf eine persönliche Merkliste setzen. Diese Funktion wird durch das Entity *WatchlistEntry* abgebildet, das Verweise auf die jeweilige Nutzer:in (*WatchlistEntry.User*) und den Artikel (*WatchlistEntry.Item*) enthält.



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 *MarketplaceContext*, 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>>`).
- Strings sollen - wenn nicht anders angegeben - mit einer Maximallänge von 255 Zeichen definiert werden.
- *User.Username* soll maximal 32 Stellen lang sein.
- Der Preis in *Item* soll mit 9 Stellen (5 Vorkomma- und 4 Nachkommastellen) gespeichert werden. Sie können das Attribut `[Precision(9, 4)]` verwenden.
- Das Attribut *Name* soll in *User* als *value object* definiert werden.
- Verwenden sie eigene primary keys mit dem Namen *Id* (autoincrement), außer im Modell ist mit *PK* explizit ein Schlüssel mit `<<PK>>` angegeben.
- Speichern Sie die enum *ItemCondition* in *Item* als String in der Datenbank ab.

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.

- PersistEnumInItemTest beweist, dass die Enum *Condition* in *Item* korrekt (als String) gespeichert wird.
- PersistValueObjectInUserTest beweist, dass Sie ein Entity vom Typ *User* mit dem Namen als value object als value object speichern können.
- EnsureNameInCategoryIsUniqueTest beweist, dass Sie nur ein Entity vom Typ *Category* mit gleichem Namen speichern können.

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

Bewertung (gesamt: 33 Punkte, 36%)

Aufgabe 1 (33 Punkte in Summe)	**Ges**
Das erstellte Domainmodel kann vom OR Mapper erzeugt werden.	3
Das Entity User wird korrekt in der erzeugten Datenbank abgebildet.	3
User.Name ist als primary key definiert.	2
Das Entity Category wird korrekt in der erzeugten Datenbank abgebildet.	3
Category.Name ist als unique definiert.	2
Das Entity Item wird korrekt in der erzeugten Datenbank abgebildet.	3
Die Enum Item.Condition wird als String in der Datenbank gespeichert.	2
Das Entity Message wird korrekt in der erzeugten Datenbank abgebildet.	3
Das Entity WatchlistEntry wird korrekt in der erzeugten Datenbank abgebildet.	3
Der Test EnsureNameInCategoryIsUniqueTest ist korrekt (Aufbau, Durchlauf)	3

Aufgabe 1 (33 Punkte in Summe)	Ges
Der Test PersistEnumInItemTest ist korrekt (Aufbau, Durchlauf)	3
Der Test PersistValueObjectInUserTest ist korrekt (Aufbau, Durchlauf)	3

Teilaufgabe 2: Abfragen und Servicemethoden

Das folgende Modell beschreibt ein System zur Verwaltung eines Essens-Lieferdienstes.

Kund:innen müssen sich im System registrieren, um Bestellungen aufgeben zu können. Ihre Kontaktdaten werden im Entity *Customer* gespeichert. Es enthält Informationen wie *Customer.PhoneNumber* und *Customer.Email*.

Restaurants, die am Lieferservice teilnehmen, sind im Entity *Restaurant* abgebildet. Es enthält den Namen (*Restaurant.Name*) und die Adresse (*Restaurant.Address*). Jedes Restaurant bietet eine Auswahl an Produkten (*Product*) an, die über das System bestellt werden können.

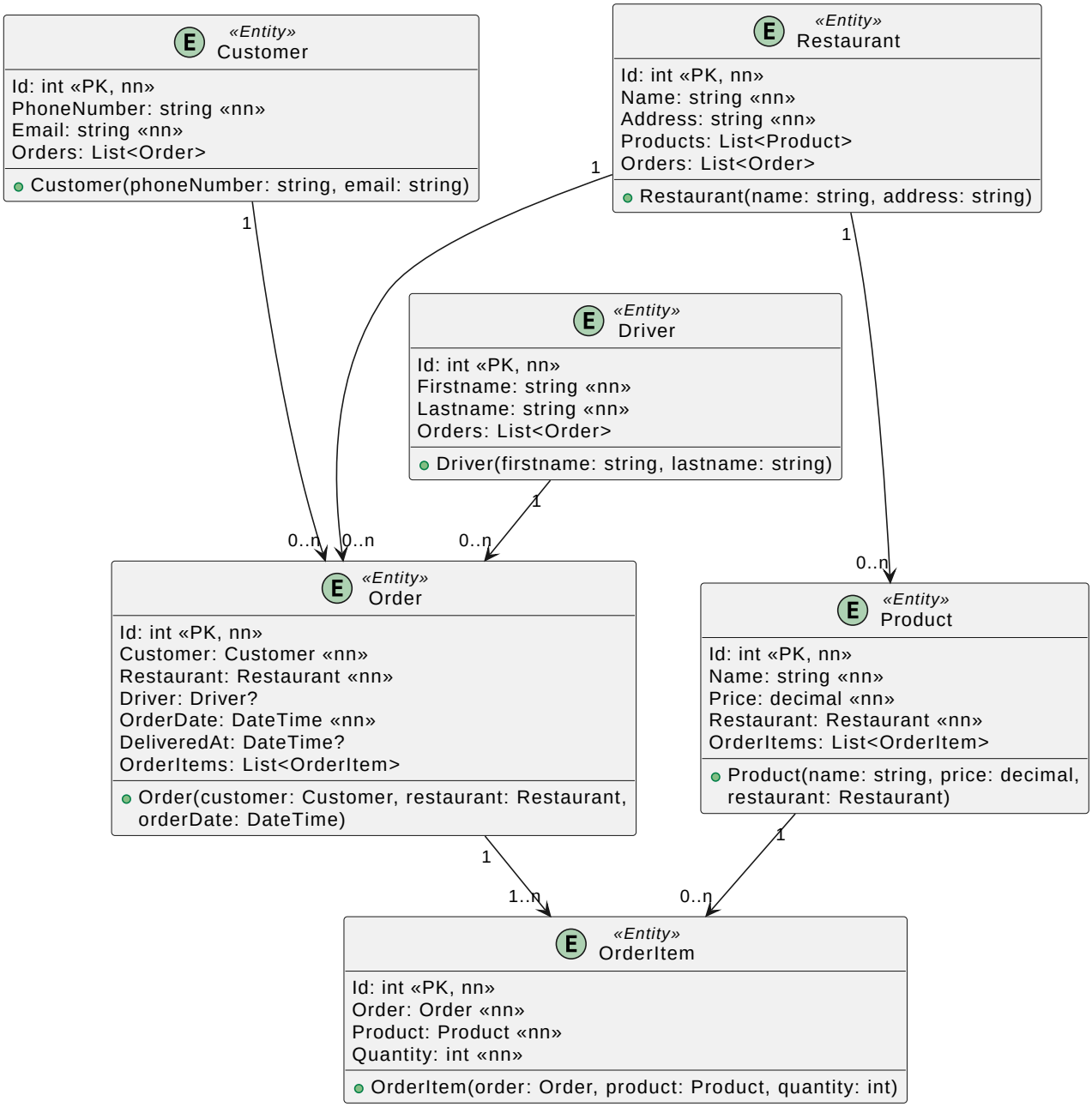
Die angebotenen Speisen oder Getränke sind im Entity *Product* gespeichert. Jedes Produkt hat einen Namen (*Product.Name*), einen Preis (*Product.Price*) und gehört genau zu einem Restaurant (*Product.Restaurant*).

Kund:innen können bei einem Restaurant eine Bestellung aufgeben. Diese wird im Entity *Order* erfasst. Eine Bestellung ist immer einem Kunden (*Order.Customer*) und einem Restaurant (*Order.Restaurant*) zugeordnet. Optional kann auch ein:e Fahrer:in (*Order.Driver*) zugewiesen sein, die bzw. der die Lieferung übernimmt. Die Bestellung speichert außerdem das Bestelldatum (*Order.OrderDate*) sowie optional den Zeitpunkt der Lieferung (*Order.DeliveredAt*). Ist das Feld *Order.DeliveredAt* null, so weiß das System, dass die Bestellung noch nicht zugestellt wurde.

Die einzelnen Positionen einer Bestellung sind im Entity *OrderItem* gespeichert. Ein *OrderItem* ist jeweils einem Produkt (*OrderItem.Product*) und einer Bestellung (*OrderItem.Order*) zugeordnet und enthält die Menge (*OrderItem.Quantity*).

Die Fahrer:innen, die die Bestellungen ausliefern, sind im Entity *Driver* gespeichert. Es enthält den Vornamen (*Driver.Firstname*) und Nachnamen (*Driver.Lastname*). Ein:e Fahrer:in kann mehreren Bestellungen zugeordnet sein.

Das folgende Modell zeigt die im Projekt vorhandenen Entities.



Arbeitsauftrag

In den Dateien *src/SPG_Fachtheorie.Aufgabe2.Base/Infrastructure/DeliveryContext.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/DeliveryService.cs* durch. Verwenden Sie den über Dependency Injection bereitgestellten *DeliveryContext*.

public List<CustomerDto> GetAllCustomers()

Diese Methode gibt eine Liste aller registrierten Kunden zurück. Die Daten stammen aus dem Entity *Customer*.

Als Rückgabetyp steht der folgende Record zur Verfügung:

```
public record CustomerDto(int CustomerId, string Email, string PhoneNumber);
```

CustomerId: ID des Kunden, *Email*: E-Mail-Adresse des Kunden, *PhoneNumber*: Telefonnummer des Kunden.

public List<OrderSummaryDto> GetOrdersByCustomer(int customerId)

Diese Methode gibt alle Bestellungen zurück, die ein bestimmter Kunde (*customerId*) aufgegeben hat. Die Daten stammen aus dem Entity *Order*.

Als Rückgabetyp steht der folgende Record zur Verfügung:

```
public record OrderSummaryDto(
  int OrderId, int RestaurantId, string RestaurantName,
  DateTime OrderDate, DateTime? DeliveredAt);
```

OrderId: ID der Bestellung, *RestaurantId*: ID des Restaurants, *RestaurantName*: Name des Restaurants, *OrderDate*: Zeitpunkt der Bestellung, *DeliveredAt*: Zeitpunkt der Lieferung (falls bereits geliefert, sonst null).

public List<ProductSalesDto> GetProductsWithRevenue()

Diese Methode gibt alle Produkte samt ihrem Umsatz zurück. Der Umsatz ergibt sich aus *Product.Price* × *OrderItem.Quantity* und wird über alle Bestellungen summiert.

Auch Produkte, die bisher nicht bestellt wurden, werden mit einem Umsatz von *0.0* (decimal) gelistet.

Als Rückgabetyp steht der folgende Record zur Verfügung:

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

ProductId: ID des Produkts, *ProductName*: Name des Produkts, *Revenue*: Gesamtumsatz des Produkts.

public List<DriverDeliveryCountDto> GetDriverDeliveryCounts()

Diese Methode liefert eine Liste aller Fahrer:innen mit der Anzahl der Bestellungen, die sie ausgeliefert haben. Die Anzahl wird über alle *Order*-Einträge gezählt, bei denen der Fahrer eingetragen ist.

Als Rückgabetyp steht der folgende Record zur Verfügung:

```
public record DriverDeliveryCountDto(
  int DriverId, string Firstname, string Lastname, int DeliveryCount);
```

DriverId: ID des Fahrers, *Firstname*, *Lastname*: Name des Fahrers, *DeliveryCount*: Anzahl der ausgelieferten Bestellungen

public Order PlaceOrder(int customerId, int restaurantId, Product product, int quantity)

Diese Methode legt eine neue Bestellung (*Order*) für einen bestimmten Kunden in einem bestimmten Restaurant an. Die Bestellung enthält genau ein Produkt mit der gewünschten Menge (*quantity*). Es wird auch ein *OrderItem* erstellt.

Folgende Randbedingungen gelten:

- Ist *quantity* kleiner oder gleich 0, wird eine *DeliveryServiceException* mit dem Text *Quantity must be positive.* geworfen.
- Wird *customerId* nicht in *Customers* gefunden, ist eine *DeliveryServiceException* mit dem Text *Customer not found.* zu werfen.
- Wird *restaurantId* nicht in *Restaurants* gefunden, ist eine *DeliveryServiceException* mit dem Text *Restaurant not found.* zu werfen.
- Bietet das Restaurant das angegebene Produkt nicht an, wird eine *DeliveryServiceException* mit dem Text *Restaurant does not offer this product.* geworfen.

Als Rückgabewert wird das erzeugte *Order*-Entity zurückgegeben.

public void MarkOrderAsDelivered(int orderId)

Diese Methode markiert eine Bestellung als ausgeliefert. Der Zeitstempel *Order.DeliveredAt* wird auf den aktuellen Zeitpunkt gesetzt (*DateTime.UtcNow*).

Folgende Randbedingungen gelten:

- Wird *orderId* nicht gefunden, ist eine *DeliveryServiceException* mit dem Text *Order not found.* zu werfen.
- Ist die Bestellung bereits als geliefert markiert (*DeliveredAt* ist nicht *null*), so ist eine *DeliveryServiceException* mit dem Text *Order already delivered.* zu werfen.

Testen Ihrer Implementierung

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

Bewertung (gesamt: 26 Punkte, 28%)

Aufgabe 2 (26 Punkte in Summe)	Ges
Die Methode GetAllCustomers ist korrekt.	2
Die Methode GerOrdersByCustomer ist korrekt.	2
Die Methode GetProductsWithRevenue ist korrekt.	3
Die Methode GetDriverDeliveryCounts ist korrekt.	2
Die Methode PlaceOrder berücksichtigt "Quantity must be positive".	2
Die Methode PlaceOrder berücksichtigt "Customer not found".	2
Die Methode PlaceOrder berücksichtigt "Restaurant not found.".	2
Die Methode PlaceOrder berücksichtigt "Restaurant does not offer this product.".	2
Die Methode PlaceOrder arbeitet unter Regelbedingungen korrekt.	3
Die Methode MarkOrderAsDelivered berücksichtigt "Order not found".	2
Die Methode MarkOrderAsDelivered berücksichtigt "Order already delivered".	2
Die Methode MarkOrderAsDelivered arbeitet unter Regelbedingungen korrekt.	2

Teilaufgabe 3: REST(ful) API

Für das vorige Modell des Essens-Lieferdienstes 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 Controller bereit. Über Dependency Injection wird ein mit Musterdaten gefüllter *DeliveryContext* bereitgestellt. Führen Sie alle Datenbankabfragen nicht blockierend mit *await* und *async* durch. Verwenden Sie im Fehlerfall die Methode *Problem()* mit geeignetem HTTP-Statuscode, um RFC-9457-konforme Antworten zu liefern.

GET /customers

Diese REST API Route liefert eine Liste aller registrierten Kunden zurück.

Als Rückgabetyp steht Ihnen folgender DTO-Record zur Verfügung:

```
public record CustomerDto(int Id, string PhoneNumber, string Email);
```

Id: ID des Kunden, *PhoneNumber*: Telefonnummer des Kunden, *Email*: E-Mail-Adresse des Kunden.

Table 1. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
200	Erfolgreiche Rückgabe der Liste aller Kunden als Array von *CustomerDto*-Objekten.

Response für GET /customers

```
[
  {
    "id": 1,
    "phoneNumber": "(0818) 549603751",
    "email": "Thalea62@gmail.com"
  },
  {
    "id": 2,
    "phoneNumber": "+49-873-2641889",
    "email": "Bilal98@hotmail.com"
  },
  {
    "id": 3,
    "phoneNumber": "+49-2325-30053148",
    "email": "Collien_Isekenmeier@yahoo.com"
  },
  {
    "id": 4,
    "phoneNumber": "+49-747-6533606",
    "email": "Marcus.Kolokas@gmail.com"
  },
  {
    "id": 5,
    "phoneNumber": "(0758) 027678403",
    "email": "Luke.Mesloh@yahoo.com"
  }
]
```

GET /customers/{id}/orders?includeDelivered=(bool)

Diese Route liefert die Informationen eines bestimmten Kunden inklusive seiner Bestellungen zurück. Optional kann per Query-Parameter *includeDeliverede* auch die Zustellung bereits abgeschlossener Bestellungen berücksichtigt werden. Wird er Parameter *includeDeliverede* nicht angegeben, so ist er *false*, d. h. es werden nur die noch nicht zugestellten Bestellungen zurückgegeben. Eine Bestellung gilt als zugestellt (delivered), wenn das Feld *Order.DeliveredAt* nicht *null* ist. Als Rückgabetyp steht Ihnen folgender DTO zur Verfügung:

```
public record CustomerWithOrdersDto(int Id, string Email, List<OrderDto> Orders);
public record OrderDto(int OrderId, int RestaurantId, string RestaurantName, DateTime OrderDate,
DateTime? DeliveredAt);
```

Id: ID des Kunden, *Email*: E-Mail-Adresse, *Orders*: Liste der zugehörigen Bestellungen, ** *OrderId*: ID der Bestellung

RestaurantId: ID des Restaurants, *RestaurantName*: Name des Restaurants, *OrderDate*: Datum der Bestellung, *DeliveredAt*: Datum der Zustellung, wenn bereits geliefert, sonst null.

Table 2. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
200	Erfolgreiche Rückgabe der korrekt gefilterten Daten in einem *CustomerWithOrdersDto*-Objekt ohne gesetztem includeDelivered Parameter.
200	Erfolgreiche Rückgabe der korrekt gefilterten Daten in einem *CustomerWithOrdersDto*-Objekt mit gesetztem includeDelivered Parameter.
404	Wenn kein Kunde mit der angegebenen ID existiert.

Response für GET /customers/3/orders?includeDelivered=false


```
{
  "id": 3,
  "email": "Collien_Isekenmeier@yahoo.com",
  "orders": [
    {
      "id": 8,
      "restaurantId": 2,
      "restaurantName": "Saflanis - Rahn",
      "orderDate": "2025-06-19T18:14:03",
      "deliveredAt": null
    }
  ]
}
```


Response für GET /customers/3/orders?includeDelivered=true


```
{
  "id": 3,
  "email": "Collien_Isekenmeier@yahoo.com",
  "orders": [
    {
      "id": 5,
      "restaurantId": 1,
      "restaurantName": "Thomas - Hengmith",
      "orderDate": "2025-01-23T01:52:49",
      "deliveredAt": "2025-01-23T02:45:49"
    },
    {
      "id": 6,
      "restaurantId": 1,
```

```
      "restaurantName": "Thomas - Hengmith",
      "orderDate": "2025-01-06T17:35:24",
      "deliveredAt": "2025-01-06T18:03:24"
    },
    {
      "id": 8,
      "restaurantId": 2,
      "restaurantName": "Saflanis - Rahn",
      "orderDate": "2025-06-19T18:14:03",
      "deliveredAt": null
    }
  ]
}
```


PATCH /customers/{id}

Diese Route aktualisiert das den Datensatz eines Kunden. *{id}* ist die Customer ID. Die neuen Werte für Telefonnummer und E-Mail-Adresse werden über den request body übergeben. Beachten Sie, dass die Stringlänge von *PhoneNumber* und *Email* mindestens 1 Stelle lang sein muss.

Implementieren Sie zuerst den Record in *SPG_Fachtheorie_Loesung/src/SPG_Fachtheorie.Aufgabe3/Commands/UpdateCustomerProfile.cs* mit den entsprechenden Properties und Validierungen.

Table 3. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
204	Erfolgreiches Update des Kundenprofils
400	Fehler beim Speichern, z. B. bei ungültiger Eingabe oder Datenbankfehler
404	Wenn kein Kunde mit der angegebenen ID gefunden wurde.

Verfassen von Integration Tests

In *test/SPG_Fachtheorie.Aufgabe3.Test/CustomerControllerTests.cs* sollen Integration Tests verfasst werden. Es soll **nur der Endpunkt GET /customers/{id}/orders** geprüft werden. Prüfen Sie 3 Bedingungen:

- Der verfasste Integrationtest zeigt, dass *GET /customers/{id}/orders* bei ungültiger ID HTTP 404 liefert.
- Der verfasste Integrationtest zeigt, dass *GET /customers/{id}/orders* ohne Filter korrekte Daten liefert.
- Der verfasste Integrationtest zeigt, dass *GET /customers/{id}/orders* mit Filter korrekte Daten liefert.

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*.

Bewertung (gesamt: 33 Punkte, 36%)

GET /customers liefert HTTP 200.	**1**
GET /customers liefert korrekte Daten.	3
GET /customers/{id}/orders liefert HTTP 200 bei gültiger ID.	1
GET /customers/{id}/orders liefert HTTP 404 bei ungülltiger ID.	1
GET /customers/{id}/orders liefert korrekte Daten ohne Filterparameter.	3
GET /customers/{id}/orders liefert korrekte Daten bei includeDelivered=false	3
GET /customers/{id}/orders liefert korrekte Daten bei includeDelivered=true	3
PATCH /customers/{id} liefert HTTP 404 bei ungültiger ID.	1
PATCH /customers/{id} liefert HTTP 400 bei ungültiger PhoneNumber.	2
PATCH /customers/{id} liefert HTTP 400 bei ungültiger Email.	2
PATCH /customers/{id} liefert HTTP 204 bei gültiger ID.	1
PATCH /customers/{id} liefert HTTP 204 aktualisiert korrekt die Datenbank unter Regelbedingungen.	3
Der verfasste Integrationtest zeigt, dass GET /customers/{id}/orders bei ungültiger ID korrekt arbeitet.	3
Der verfasste Integrationtest zeigt, dass GET /customers/{id}/orders ohne Filter korrekt arbeitet.	3
Der verfasste Integrationtest zeigt, dass GET /customers/{id}/orders mit Filter korrekt arbeitet.	3

Wichtiger Hinweis nach Arbeitsende

Schließen Sie Visual Studio. Starten Sie das Skript **compile.cmd** im Ordner *SPG_Fachtheorie*. Es kontrolliert, ob Ihre Projekte kompiliert werden können. **Projekte, die nicht kompilieren, können nicht bewertet werden!**